NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Balázsi Edina**

2022 Hallgató törzskönyvi száma: **T/004564/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Biomatika és Alkalmazott Mesterséges Intelligencia Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve:	**Balázsi Edina**
Törzskönyvi száma:	T/004564/FI12904/N
Neptun kódja:	[redacted]

A dolgozat címe:

Blockly alapú virtuális robot programozási környezet továbbfejlesztése
Development of a Blockly based virtual robot programming environment

Intézményi konzulens:	Dr. Galambos Péter
Külső konzulens:	Budai Tamás (MISTEMS Kft.)
Beadási határidő:	2021. december 15.
A záróvizsga tárgyai:	Számítógép architektúrák IoT, beágyazott rendszerek és robotika specializáció

A feladat

A különféle robotalkalmazások egyre inkább megjelennek a mindennapjainkban, így a robotikai alapismeretek szükségszerűen az alapvető műveltség részévé válnak. Világszerte több vállalat és kutatóhely dolgozik azon, hogy a robotok programozását és általában a robotika lehetőségeinek megismerését lehetővé tegyék az iskolás korú gyermekek számára. Egyik fontos megközelítés a blokk alapú robotprogramozás, ami egyszerű vizuális szerkesztő felülettel támogatja a programalkotást. Ez a módszer már bizonyított a gyermekek programozás oktatása során.

A MISTEMS Kft. az Óbudai Egyetemmel együttműködve a MaxWhere virtuális valóság környezetben alakít ki olyan robotprogramozási környezetet, amelyben a népszerű Universal Robots robotkarokat lehet blokk alapú megközelítésben programozni.

A dolgozat célja a Roblockly környezet prototípusának továbbfejlesztése és egy következő fejlettségi szintre való eljuttatása.

A dolgozatnak tartalmaznia kell:

- a vizuális robotprogramozási környezetek áttekintését internetes források és a szakirodalom alapján,
- a Roblockly rendszer kiindulási állapotának ismertetését,
- az alkalmazott technológiák bemutatását,
- a továbbfejlesztés során megcélzott részproblémák ismertetését,
- a fejlesztés rendszertervét,
- az implementáció részleteit,
- felhasználói és fejlesztői dokumentációt,
- az eredmények értékelését.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest Ph.

.....
Dr. Eigner György
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

Budai B.
.....
külső konzulens

.....
intézményi konzulens

NEUMANN JÁNOS
INFORMATIKAI KAR

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat/diplomunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladat kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. 05. 12.

Baldaszti Edina

hallagtó aláírása

r

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Balázsi Edina

Neptun kód: [redacted]

Tagozat: Nappali

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Blockly alapú virtuális robot programozási környezet továbbfejlesztése

Szakdolgozat / Diplomamunka[2] címe angolul:

Development of a Bockly based virtual robot programming environment

Intézményi konzulens: Dr. Galambos Péter

Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.02.17.	Irodalomkutatás módszertanának megbeszélése	
2.	2021.03.23.	Az első félvére vonatkozó tartalmi keretek meghatározása	
3.	2021.04.06.	Fejlesztési célok kitűzése	
4.	2021.05.11.	Beadás előtti korrekciók	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021. 05. 14.

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	**Neptun kód:**	**Tagozat:**
Balázsi Edina	[redacted]	Nappali

Telefon: **Levelezési cím (pl: lakcím):**

[redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Blockly alapú virtuális robot programozási környezet továbbfejlesztése

Szakdolgozat / Diplomamunka[2] címe angolul:

Development of a Blockly based virtual robot programming environment

Intézményi konzulens:	**Külső konzulens:**
Dr. Galambos Péter	Budai Tamás (MISTEMS Kft.)

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	**Dátum**	**Tartalom**	**Aláírás**
1.	2022.02.10.	A teszt szerver kialakítása	
2.	2022.03.03.	A CI/CD pipeline	
3.	2022.04.07.	Felmerült kérdések megvitatása	
4.	2022.04.28.	Szakdolgozat bemutatása	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022. 05. 13.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

Tartalmi összefoglaló **1**

Abstract **2**

A megoldandó probléma megfogalmazása **3**

1. Bevezetés **4**

2. Vizuális programozási nyelvek **6**
2.1. Viselkedés alapú programozás 6
2.2. Esemény alapú programozás 7
2.3. Csomópont alapú programozás 7
2.4. Blokk alapú programozás 8

3. Ipari robotok programozása **9**
3.1. A technológiai eszközök történeténetének áttekintése 9
3.2. A vizuális robotprogramozási környezetek áttekintése 11
3.2.1. ABB RobotStudio - Wizard easy programming 11
3.2.2. Roxy Robotics 16
3.2.3. Ready Robotics 17
3.2.4. Drag&bot 18

4. A Roblockly rendszer architektúrája és technológiai eszközeinek áttekintése 19
4.1. A meglévő rendszer architektúrája 21
4.1.1. A Roblockly erőforrás menedzser 23
4.1.2. A Capsule szimulációs komponens 24
4.1.3. A Roblockly API bemutatása 25
4.1.4. Az NginX szerepe 26
4.1.5. Traefik 27
4.1.6. A komponensek közötti kommunikáció felépítése 28
4.2. A Roblockly további technológiai eszközeinek bemutatása 30
4.2.1. Docker 30
4.2.2. Docker Compose 31
4.2.3. AWS 33

5. A lokális teszt környezet kiépítése **34**

5.1. Felmerült problémák és megoldásaik 34

6. A CI/CD pipeline megvalósítása 38
6.1. A fejlesztés rendszerterve 41
6.2. A multi-projekt pipeline implementálása 41
6.3. Az implementáció során felhasznált technológiai eszközök 43
6.3.1. A GitLab Registry 43
6.3.2. A GitLab Runner 44
6.3.3. A GitLab Executor 45
6.3.4. A GitLab CI/CD változók 46
6.4. Az implementáció során felmerült problémák és megoldásuk 47

7. Továbbfejlesztési lehetőségek 49

8. Összegzés 50

9. Köszönetnyilvánítás 50

Hivatkozások 51

Ábrajegyzék 55

10. Mellékletek, felhasználói és fejlesztői dokumentációk 56
10.1. A lokális környezet kialakítása teszteléshez virtuális gépen 56
10.1.1. Ajánlott rendszer előkövetelmények 56
10.1.2. A Docker telepítése 56
10.1.3. A Docker-Compose telepítése 57
10.1.4. Docker képfájlok létrehozása 57
10.1.5. Docker hálózat létrehozása 58
10.1.6. A rendszer elindítása docker-compose segítségével 58
10.1.7. A host portszámának elérhetővé tétele docker konténerből 58
10.2. A GitLab CI/CD pipeline-hoz szükséges eszközök telepítése és konfigurálása 58
10.2.1. A GitLab Runner telepítése 58
10.2.2. A GitLab Runner regisztrálása 59
10.3. A Cron segédprogram konfigurációja 60
10.3.1. Előkövetelmények 60
10.3.2. A Cron feladat beállítása 60

Tartalmi összefoglaló

A robotika és a robotprogramozás meghatározó szerepet töltenek be a mindennapjainkban, ezáltal egyre inkább az alapműveltség részévé válik ezen alkalmazások használatának készsége is. Számos törekvés van arra, hogy egészen fiatal kortól kezdve, akár iskolás gyerekeket is bevezessenek a robotika világába, akik ezáltal elsajátíthatják a későbbi tanulmányaikhoz és munkájukhoz szükséges alapismereteket.

Erre a célra az egyik legalkalmasabb eszköz a blokk alapú robotprogramozás, amely egy egyszerű és látványos vizuális szerkesztő felületet biztosít a felhasználók számára. Ennek hatékonysága nem csupán a gyerekek programozás oktatása, hanem az ipari rendszerek használatának során is széles körben elismert.

Napjainkban a gyártásevolúció jelenlegi lépcsőfokán, az ipar 5.0 küszöbén a vállalatok igyekeznek nagyobb hangsúlyt fektetni az ember-robot kapcsolat fejlesztésére a minél hatékonyabb termelés kialakítása érdekében. A munkavállalóknak emiatt elengedhetetlen, hogy naprakész szakismerettel rendelkezzenek az új robottechnológiák használatát illetően. Emiatt megnövekedett az igény az univerzális célú robotprogramozó alkalmazások fejlesztésére is, amelyek között számos blokk alapú megoldás is jelen van a piacon.

Szakdolgozatom témája az Óbudai Egyetem és a MISTEMS Kft. közreműködése során fejlesztett Roblockly alkalmazás részletes bemutatása és a piacon jelenlévő hasonló megoldásokkal való összehasonlítása, valamint a szoftver egy következő fejlettségi szintre való eljuttatása. A megvalósítandó feladat célkitűzése a verziókövetés megfelelő kialakítása és az automatizált üzembe helyezés mellett egy olyan átfogó leírás biztosítása, amely jelentősen megkönnyíti a Roblockly környezet jelenlegi és jövőbeli fejlesztőinek mindennapos munkáját.

Abstract

Robotics and robot programming play an important role in our everyday lives, and the ability to use these technologies are increasingly becoming part of basic literacy. Many efforts are being made to introduce children to robotics from a very young age, even at school, so that they can acquire the basic skills they need for their future studies and work.

One of the most suitable tools for this purpose is block-based robot programming, which provides users with a simple and attractive visual editing interface. Its benefits are not only outstanding in teaching children, but are also widely recognised in the commissioning and control of industrial robots.

Nowadays, humanity is at the current stage of manufacturing evolution, on the threshold of Industry 5.0 and thus companies are trying to put more emphasis on improving the human-robot relationship in order to achieve more efficient production. It is therefore essential for employers to ensure that their employees can quickly and easily acquire up-to-date skills in the use of new robotic technologies. This has led to an increased demand for the development of general-purpose robot programming applications, including a number of block-based solutions on the market.

My thesis is about to present in detail the Roblockly application developed in collaboration with the Óbuda University and MISTEMS Kft. and to compare it with similar solutions on the market and to bring the software to the next level of development. The objective of the deliverable is to provide a comprehensive specification that will significantly facilitate the day-to-day work of current and future developers of the Roblockly environment, in addition to the proper design of version tracking and automated deployment.

A megoldandó probléma megfogalmazása

A Roblockly alkalmazás a MaxWhere virtuális valóság környezetében kialakított fejlesztői környezet, amelyben az Universal Robots ipari robotkarjait vizualizálja a virtuális térben. A robotkarokat egy blokk alapú robotprogramozási felületen képes a felhasználó az igényeinek és elképzeléseinek megfelelően programozni, és ezzel párhuzamosan a robot mozgását valós időben megfigyelni. Dolgozatom célja a prototípus egy következő fejlettségi szintre való eljuttatása a verziókövetés, illetve a CI/CD koncepciók kódbéli manifesztációjának kiépítésével, emellett kitérek a Roblockly környezet ismertetésére, elsősorban a szerver oldali részre fektetve a hangsúlyt. A Roblockly projekten Peller Gáborral közreműködve dolgoztunk együtt, akinek a szakdolgozatában[1] megfogalmazott feladata a kliens oldal korszerűsítésére terjedt ki. Munkája során a rendszer komponensei közül a webalkalmazás felhasználói felületét váltotta ki a MaxWhere-ben való vizualizáció megvalósításával. A mikroszerviz architektúra jellegéből adódóan lehetővé tette számunkra, hogy egymástól függetlenül dolgozzunk az adott komponenseken, ezáltal nem befolyásoltuk közvetlenül a másik előrehaladását.

A megoldandó feladat jelentősége kiemelten fontos a megfelelő kódbázis kiépítésének aspektusából. Egy hosszútávú projekt esetén a fejlesztői csapat számára elengedhetetlen a napi teendők gördülékeny elvégzéséhez és a közös munka támogatásához a megfelelő verziókövetés, illetve az automatizált folyamatok alkalmazása. A transzparens működés, a precíz és pontos dokumentáció, illetve a megoldások nyomon követhetősége az üzleti folytonosság megőrzése szempontjából elengedhetetlen. Ennek köszönhetően az egyes folyamatok könnyedén reprodukálhatóak, mindezek elősegítik az új alkalmazottak betanítását is. A vállalat szemszögéből kiemelten fontos, hogy a szoftver életciklus menedzsmentje leváljon a napi működésről, illetve folyamatosan kézben tartható legyen, nem csupán deep tech[2] szakértők által.

Mindemellett dolgozatomban kitérek a témához kapcsolódó szakirodalmakra és fogalmakra, illetve összevetem a piacon napjainkban jelenlévő ipari robotok vizuális programozására szánt rendszerekkel, szem előtt tartva a meglévő alkalmazás tulajdonságait.

[1] Peller Gábor: Interaktív robot programozás oktató felület továbbfejlesztése
[2] Deep tech: tudományos áttöréssel járó mérnöki innovációkra épülő technológiák összessége [1]

1. Bevezetés

Az ipar 4.0 napjaink meghatározó irányzata az ipari gyártástechnológiák szempontjából, különösen a gyártási folyamatok automatizálása és az információáramlás terén. Ez a digitális transzformáció az ipari értéklánc egy új állapotát képviseli annak megszervezésében és irányításában.

A negyedik ipari forradalom alapját képező kiberfizikai rendszerek[3] összeköttetésére használt IoT, vagyis a dolgok internete[4] segítségével a beágyazott rendszerek közti kommunikáció valósítható meg, amely hozzájárul a gyorsabb és megbízhatóbb információáramlás kialakításához az ipari folyamatláncok egyes tagjai között. A hálózat kiépítése lehetővé teszi új technikák és módszerek megvalósítását mind a termelésben, értékteremtésben, mind a valós idejű optimalizálásban egyaránt. Az úgynevezett okos gyár alapjait a felhőalapú és kognitív számítástechnika, illetve a fentebb említett kiberfizikai rendszerek és az ezt összefogó IoT megoldások képezik [2].

Számos vállalat jelenleg is ezen módszer kialakítására fókuszál, azonban - habár még kezdetleges fázisban - már megjelent az ipar 5.0 fogalma is a köztudatban. Ez a legújabb nagy hullám az ipari forradalmak tekintetében, ami magában foglalja az ember-robot kapcsolat fejlesztésének igényét a munkafolyamat során, hiszen még mindig vannak olyan munkafázisok, ahol az emberi jelenlét nem pótolható [3]. A gyártási környezetekben a robotok jellemzően az emberek számára veszélyes, fizikailag megerőltető vagy monoton feladatokat látnak el, mint például autóipari gyárakban a hegesztési vagy festési munkák, illetve raktárakban a nagy tömegű anyagok szállítása és kirakodása.

Az ipar 5.0 a gyártásevolúció következő nagy korszaka, célja a hatékonyság növelése olyan módon, hogy az egyre okosabb és egyre fejlettebb kommunikációs hálózattal összekapcsolt munkatereket az emberi találékonysággal és intelligenciával minél magasabb fokú együttműködésre ösztönözze [3]. Erre nagyszerű példa a cobotok[5], azaz a kollaboratív robotok egyre jelentősebb szerepe a gyártási folyamatokban.

[3]Kiberfizikai rendszerek: Az informatikai, szoftvertechnológiai, valamint mechanikai- és elektronikai komponensek összessége, vagyis a szoftver és hardver elemek szoros együttműködésén alapuló rendszer

[4]Internet of Things (IoT) - a dolgok internete: különböző érzékelőkkel ellátott elektronikai eszközök egymás közötti kommunikációja internet alapú hálózaton

[5]Olyan robot, amely az emberrel való közös munkára lett tervezve

A jövő gyártási folyamatainak komplexitása és az egyre inkább testreszabott megoldások támogatása központi szempont az új irányzat tekintetében. Ennek ellenére nem szabad szem elől téveszteni, hogy ez az újfajta trend nem az emberi munka teljes kiváltására irányul, sokkal inkább annak támogatására. Míg a robotok a precízebb és következetesebb feladatmegoldásra képesek, az emberi kritikus és kreatív gondolkodás, valamint az alkalmazkodóképesség nem helyettesíthető gépekkel [3]. Sokan azonban jelenleg még bizalmatlanok és valamennyire tartanak attól, hogy az automatizálás és a robotok integrációja miatt elveszthetik az állásukat, vagy éppen az újonnan létrejött munkahelyeken nem tudnák megállni a helyüket a megfelelő kvalitások hiányában.

Az ipari robotok programozása általában a robotgyártó cégek által fejlesztett saját programnyelvén történik, ami számos különböző, egymással inkompatibilis megoldást eredményez. Az egységesség hiánya jelentősen megnehezíti mind a perifériagyártó cégek - mivel ennek következtében számos szabványnak kell megfelelniük -, mind az ezeket alkalmazó vállalatok működését.

Ebből kifolyólag felmerült az igény az ipari robotvezérlő programok általános célú fejlesztésére, ezáltal már számos megoldás született a folyamat hatékonyabbá tételére különböző szoftverek és platformok formájában. Az iparban jelenlévő gyors átalakulásnak köszönhetően törekednek arra is, hogy lehetőséget biztosítsanak olyan alkalmazottak átképzésére is, akik nem rendelkeznek előzetes programozási tapasztalattal, amelyre ideális megoldás a vizuális robotprogramozási eszközök igénybevétele. Használatuk nem követel meg semmilyen előzetes ismeretséget vagy tapasztalatot, ezáltal a felhasználók betanítási ideje és költségei jelentősen csökkennek.

2. Vizuális programozási nyelvek

Napjainkban, a vizuális programozási módszerek alkalmazása a mobil applikációk, játékok fejlesztése, illetve az oktatóprogramok megvalósítása mellett az ipari robotok programozása során is felmerül lehetséges fejlesztői megoldásként. Ennek következtében egy gyorsabban implementálható és hibatűrő robotprogramot hozhat létre a felhasználó, ugyanis a vizuális programozás során nem szükséges programkódot írni: a fejlesztési folyamat grafikus felületen történik, ahol előre definiált elemekből és utasításcsomagokból építhető fel a végső program. A felhasználók így gyorsabban betaníthatóak a robotok felprogramozására, illetve komolyabb programozói tudást sem igényel ezen módszerek elsajátítása. Ezáltal a cégek optimalizálhatják a gyártási folyamatok idejét is különböző termék életciklus menedzsment[6] módszerek használata mellett. A vizuális nyelvek négy csoportba oszthatóak a felület és a kód grafikus megjelenésének kialakításától függően [4] [5]:

- Viselkedés alapú (Behavior based)
- Esemény alapú (Event-sheet based)
- Csomópont/folyamat alapú (Node based/Flow based)
- Blokk alapú (Block based)

2.1. Viselkedés alapú programozás

A viselkedés alapú programozás az egyik legegyszerűbben és leggyorsabban alkalmazható módszer a fentebb felsoroltak közül, ugyanis előre megszabott viselkedési mintákat lehet hozzárendelni a különböző objektumokhoz. Tulajdonképpen különböző forgatókönyveket definiálhatunk, amelyek meghatározzák, pontosan milyen szabályok szerint működhet a rendszer [6]. Egy egyszerűbb példán keresztül szemléltetve, a robotnak érzékelők segítségével követnie kell az előre megtervezett útvonalat és el kell kerülnie az ütközéseket. Ezen viselkedést több, kisebb objektum egymás utáni definiálásával tudjuk lemodellezni, ezáltal valamivel komplexebb feladatokat is képesek lehetünk megoldani. Habár ennek segítségével könnyebbé válik a folyamat,

[6]PLM: Product Lifecycle Management

de egyben nagyon kötötté is teszi a fejlesztés menetét, számos korlátba ütközhetünk a megvalósítás során. Viselkedés alapú programozási környezet például a LEGO MindStorms szoftvercsomag vagy a Choregraphe nevű szoftver is, ami a humanoid NAO robotok programozására szolgál.

2.2. Esemény alapú programozás

Az esemény alapú programozás már nagyobb teret enged a megvalósítás tekintetében, hiszen szinte bármilyen logika felépíthető ezen módszer alkalmazásával. A különböző entitások, úgymint az objektumok vagy szervizek egymással közvetetten, egy közvetítőn keresztül üzenetváltással kommunikálnak [7]. A fejlesztő határozhatja meg, hogy az előre megírt eseményekre a szoftver hogyan fog reagálni. Még egy egyszerű hétköznapi játék programozása során is alkalmazható ez a programozási mód, amelynek során az egyes billentyűleütések határozzák meg, hogy a karakter milyen mozdulatokat hajtson végre. Esemény alapú rendszer a Construct 2, Microsoft Visual Programming Language, vagy az ASU VIPLE is.

2.3. Csomópont alapú programozás

A csomópont vagy más néven folyamat alapú programozással az esemény alapúhoz hasonlóan számos funkció felépíthető. Ennél a rendszernél az egyes csomópontokhoz van rendelve a logika és gráfszerűen vannak kialakítva az egymás közti kapcsolatok. Az egyes folyamatok a feketedobozként működő csomópontok hálózata, amelyek előre meghatározott feladatok végrehajtásáért felelnek [8]. Az átláthatóságot azonban jelentősen megnehezítheti egy-egy feladat komplexitása, főként egy bonyolultabb, több feltételes elágazást tartalmazó program esetében, amelynek működési elve nehezen értelmezhető. Ugyanakkor a rendszer skálázhatóságának megoldása is problémás lehet ezen módszer használatakor: a különböző csomópontok és elágazások sokszorosítása egy vizuálisan nehezen értelmezhető, bonyolult architektúrát eredményezhet, amelyben a hibák megtalálása és javítása is kihívást jelent. Csomópont alapú technológia használatára példaként szolgálhat az IBM által fejlesztett Node-RED, vagy a n8n, illetve az ioBroker is, ami az IoT eszközök integrációs platformja.

2.4. Blokk alapú programozás

A blokk alapú programozás a legelterjedtebb az oktatásban, ugyanis ez hasonlít a leginkább a hagyományos értelemben vett programnyelvekhez. Az egymást követő kódblokkok sorban, egymás után teljesítődnek, ahogyan a szöveg alapú programozási nyelvek is legtöbb esetben szekvenciális utasításvégrehajtást követnek [4]. A legnépszerűbb blokk alapú grafikus programozói eszközök egyike a Scratch, az Ardublock vagy a Nepo. Emellett a Microsoft Robotics Developer Studio egy komplex fejlesztői környezetet biztosít egy Windows alapú rendszeren belül számos robot hardver programozására [9].

3. Ipari robotok programozása

3.1. A technológiai eszközök történeténetének áttekintése

Napjainkban a legtöbb robotgyártó cég rendelkezik egyedi robotprogramozási nyelvvel, melyet kifejezetten a saját termékükhöz fejlesztettek ki, azonban ez a jelenség hosszú ideje az ipari robotika egyik fő problémája. Ennek eredményeképp a robot periféria gyártóknak számos különböző protokollt kell támogatniuk vagy akár módosítaniuk kell a robot vezérlési programokat is, ami egy igen időigényes és költséges folyamat, annak ellenére, hogy plusz funkcionalitást nem ad a meglévő rendszerekhez.

Alapvetően a robotprogramozási nyelvek 30-40 évvel ezelőtti technológiákból származtathatóak, amelyek hatékonyságuk miatt azóta is töretlen népszerűséggel rendelkeznek a területen. A teljesség igénye nélkül ilyen, Pascal alapú nyelvek például az ABB RAPID programozási nyelve, a KUKA KRL, illetve a TP és a Karel nyelvek is [10] [11]. A Fanuc robotjai kezeléséhez a Teach Pendant-hoz használt TP nyelv mellett kifejlesztette az alacsonyabb szintű, Karel programnyelvet is, azonban ennek elsajátítása jóval nagyobb tapasztalatot és képzettséget követel az előbbi módszerrel szemben. Az elmúlt években több általános célú, gyártófüggetlen megoldás is született, mint például a ROS-Industrial szoftvercsomagja, amely egy szabványosabb megközelítést kínál a programozók számára egy magas szintű programnyelv segítségével [12]. Ez természetesen nem alternatívája például a KUKA KRL-nek, ami egy moduláris megoldás, amely segítségével nagyobb, bonyolultabb mozgások és rendszerek valósíthatóak meg.

Az ipari robotok kezelésére használt újhullámos technológiák közé sorolható a KUKA Sunrise rendszerszoftver, amely számos funkcióval rendelkezik a kollaboratív robotok üzemeltetéséhez, valamint folyamatos szoftverfrissítésekkel támogatják a hatékony működést [13]. Említésre méltó még a Universal Robots által fejlesztett URScript is, amely a Python programozási nyelvre hasonlít.

A legkorszerűbb műszaki megvalósítások közé sorolható a TCP/IP kapcsolatot használó módszerek, ahol a ROS környezetet ROS komponensekbe képezik le. Ilyen például a MoveIt, amely egy komplex rendszerek kialakításához használt ingyenes, nyílt forráskódú robot manipulációs platform [14] [15].

Az ipari robotok programozására kétfajta módszert különböztethetünk meg: az online programozási módot, amelynek során a robotot a gyártási folyamat leállítására késztetik, majd áttérnek a programozói módba, ahol létrehozható vagy módosítható a kód. A másik eljárás az offline programozás, amelynek során a kód megírásához nem szükséges a robot jelenléte. Amikor a program elkészül, feltöltésre kerül a robotra, emiatt a termelési folyamatot nem kell leállítani az újraprogramozás alatt [16].

Az ipari robotok vizuális programozására napjainkban egyre több megoldást találhatunk, ahol a low-code/no-code megközelítési módszer kerül középpontba. A technológia lényegében egy olyan szoftverfejlesztési megközelítést tesz lehetővé, ahol az alkalmazások és a folyamatok létrehozásához alig vagy egyáltalán nem szükséges programkód írása. A low-code fejlesztői platformok vizuális modellező eszközökkel segítik egy egyszerű ”fogd és vidd” logika alapján felépíteni a programot, egy gyors és egyszerű alternatívát biztosítva a bonyolultabb, tradicionális programkódok használatának kiváltása érdekében. A módszer tulajdonságaiból adódóan további előnyök is származhatnak, mint például az előre konfigurált modulok újrafelhasználhatósága, vagy a skálázhatóság, ami a felhasználók számának növekedése miatt lehet fontos [17].

Az előzőekben általánosságban kerültek bemutatásra az ipari robotok programozására használt különböző módszertanok. Dolgozatomban a továbbiakban olyan rendszereket járok körbe, amelyek ezen műszaki lehetőségekre kiváló példaként szolgálnak. Ezek közül az ABB, illetve a Roxy Robotics platformja hasonlít a leginkább a Roblockly szoftverhez a felhasználói interfész megvalósításának tekintetében, hiszen a négy bemutatott applikáció közül ez a kettő rendelkezik blokk alapú programozási felülettel. Emellett kitérek más cégek egyedi megközelítésére is a témához kapcsolódóan, ahol az előzőektől eltérően a rendszer más vizuális programozási eszközökön alapszik. A Ready Robotics szoftver használatakor egy folyamatábra segítségével határozhatjuk meg a robotkar működését, amely jellegét tekintve inkább a viselkedés alapú, illetve az esemény alapú programozási nyelvekhez sorolható. A Drag&bot megoldása szintén egy viselkedés alapú programozói felületet biztosít a felhasználók számára.

3.2. A vizuális robotprogramozási környezetek áttekintése

Az alábbiakban a piacon jelenlévő vizuális robotprogramozási környezetek áttekintése következik internetes források és a szakirodalom alapján. Ezen szoftverek a Roblockly alkalmazáshoz hasonló megoldásokat alkalmaznak a felhasználói felület megjelenítése szempontjából.

3.2.1. ABB RobotStudio - Wizard easy programming

A RobotStudio az ABB szimulációs és offline programozási szoftvere, a világ egyik legismertebb eszköze a robotikában. Egy teljes digitális ikerpárt szolgáltat a fizikai rendszerekhez, ezáltal távolról is szimulálható a robotcella működése, ami lehetővé teszi egy 3D-s környezetben a felhasználók számára a teljes robot installáció manipulálását anélkül, hogy a tényleges gyártósorukat meg kellene zavarni. A szoftver folyamatos fejlesztés alatt áll a hibajavítás és új funkciók hozzáadásával. Az eszköz az ABB Virtual Controller-re épül, amely a gyártósoroknál jelenlévő robotokat futtató valódi szoftver pontos másolata. Ez lehetővé teszi a minél pontosabb szimulációk elvégzését, valós robot programkódok és konfigurációs fájlok felhasználásával [18].

A cég saját fejlesztésű robotprogramozási nyelve a Rapid, ami egy magas szintű programozási nyelv az ipari robotok vezérléséhez. A szoftver teljes körű szolgáltatásokkal és kiegészítőkkel rendelkezik, amelyek lehetővé teszik az offline szimuláció megfelelő működését, csökkenti a kockázatokat és felgyorsítja az előkészítési és átállási folyamatokat a termelékenység növelése érdekében [19].

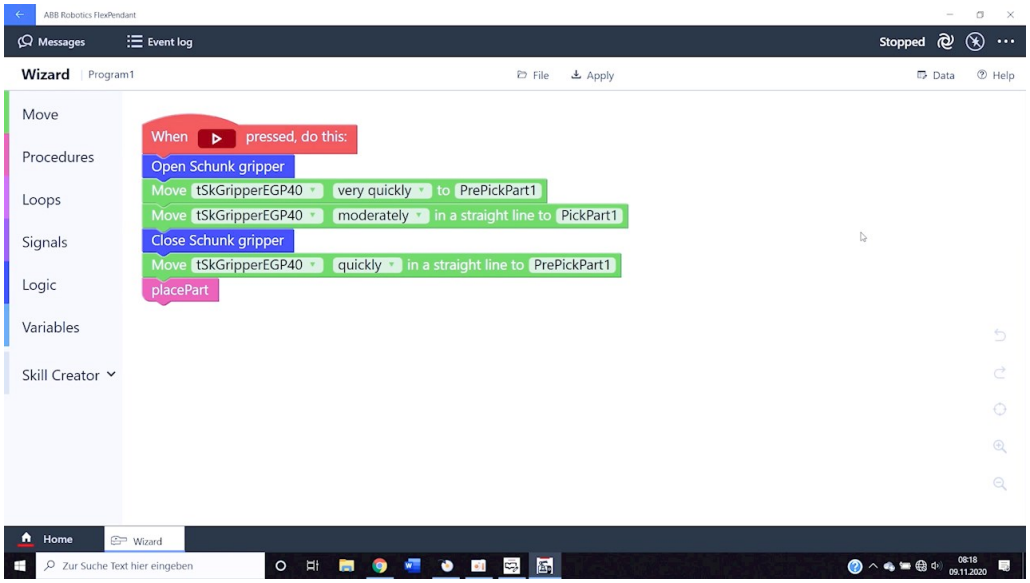
A RobotStudio legfőbb funkciói közé tartozik az úgynevezett Digital Twin koncepció, amely az automatizálási folyamatot menedzseli és optimalizálja a gyártás megzavarása nélkül. Végrehajtja a termelési rendszer valós idejű szimulációját, ami a felhasználók számára azt jelenti, hogy kipróbálhatják és tesztelhetik a módosításokat és optimalizálási folyamatokat a virtuális térben anélkül, hogy a gyártást megzavarnák [18].

Az ABB virtuális komissiózási megoldásai felgyorsítják a valós folyamatot azáltal, hogy a RobotStudio gyártási cellájának pontos másolatát használják a szimuláció során, így előre megoldható minden felmerülő technikai kérdés vagy probléma. A RobotStudio biztosítja a PLC-khez és más külső eszközökhöz való csatlakozást is, hogy a fizikai

vonal telepítése előtt a cella teljes logikáját és biztonságát tesztelni lehessen egy teljesen virtuális környezetben. Emellett egy nem mindennapi megoldást is beépítettek, ugyanis a kiterjesztett valóság technológia használatával a robotmegoldások úgy vizualizálhatóak, hogy a modellezett megoldást hologramként vetíti a valós gyártási környezetre. Ez egy AR szemüvegen keresztül a RobotStudio alkalmazásban létrehozott szimulációk vizualizálásával vagy okostelefonon, táblagépen történő applikációk használatával valósul meg [18].

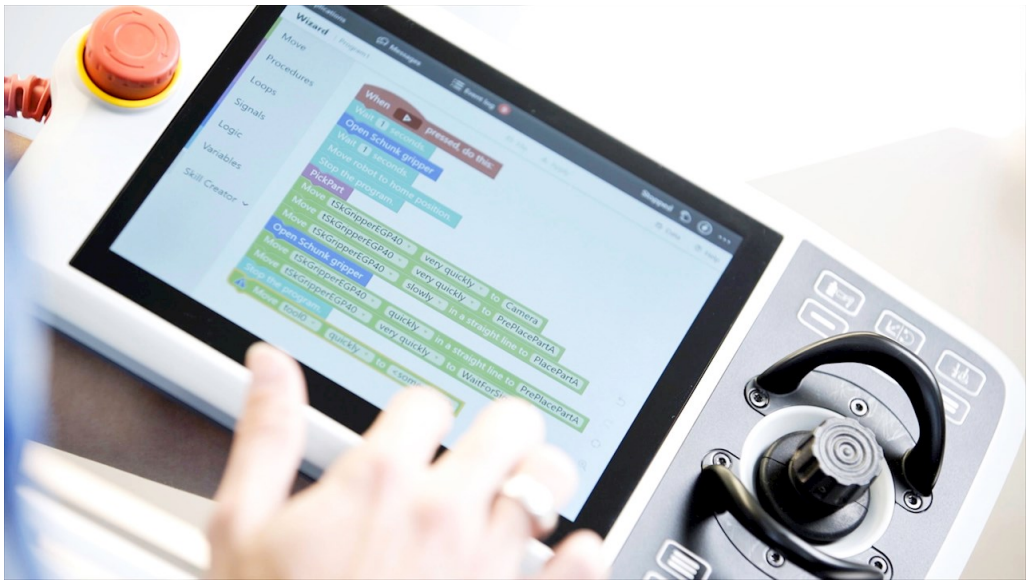
Az előbb említett szolgáltatások mellett az ABB RobotStudio kibővítésre került azzal a képességgel, hogy a SCARA robotok PC-ről is vezérelhetőek legyenek. A Robot Control Mate a RobotStudio egy kiegészítője, amely segítséget nyújt a felhasználók számára, hogy manuálisan mozgassák, betanítsák és kalibrálhassák a robotokat a számítógépről, így minden eddiginél könnyebben kezelhető egy SCARA robot mozgása. Az ABB offline programozó szoftvere az első, amely a használható a robot fizikai mozgásának valós idejű vezérlésére [20].

A RobotStudio Blockly alapú megoldása a Wizard Easy Programming alkalmazás alapját képező CoBlox robotprogramozási nyelv, ami egy blokk alapú grafikus felhasználói réteg az alap RAPID programozói réteg fölött. Ezáltal a felhasználók intuitív módon képesek ipari robotok programozására, bármilyen előzetes tudás vagy tapasztalat megléte nélkül is. A Wizard-ban létrehozott grafikus program valós időben fordul RAPID kódra, amelyet a felhasználó bármikor megtekinthet a menüben [21].



3.1. ábra. Wizard Easy Programming szoftver (forrás: https://new.abb.com/)

Az új szoftver, vagyis a Wizard Easy Programming előnye többek között az egyszerűségében rejlik, ugyanis semmilyen előzetes képzés nem szükséges az elsajátításához, használata egyszerű és az alkalmazásban található oktatóanyag segítségével gyorsan megérthető. A grafikus felhasználói interfésznek köszönhetően nem kell hagyományos kódsorokat írni, mindemellett az előkészítési folyamatok is megspórolhatók, úgymint az eszközök előzetes definiálása vagy a konfigurációk beállítása. Ehelyett a fejlesztés a ”fogd és vidd” módszer segítségével, előre definiált modulokkal történik. Ennek köszönhetően a kollaboratív és ipari robotok programozása azok számára is csupán perceket vesz igénybe, akik először használnak hasonló felületet [21].



3.2. ábra. az ABB FlexPendant felülete (forrás: https://new.abb.com/)

A Skill Creator szofver használatával akár egyedi RAPID kód alapú blokkok létrehozása is lehetséges, ennélfogva egy rugalmas és hatékony módszert biztosít a fejlesztési folyamat végrehajtásához, legyen szó speciális megfogó szerkezetekről vagy látási funkciók kivitelezéséről. Jelenleg a Wizard segítségével számos kollaboratív robot, úgymint az egykarú YuMi robotkar vagy a GoFa, illetve a SWIFTI kobotok programozását is megvalósíthatjuk. Ezenfelül ipari robotok kezelésére is használható, mint például az IRB 1100-as vagy az IRB 1300-as robot esetén. A szoftvert nem csak új hardverek mellé biztosítják, hanem a már meglévő robotokra is egyszerűen telepíthető a FlexPendant frissítésével. Abban az esetben, ha nem áll rendelkezésre fizikai robot, a Robot Studio által biztosított virtuális ikerpár programozása is lehetséges [21].

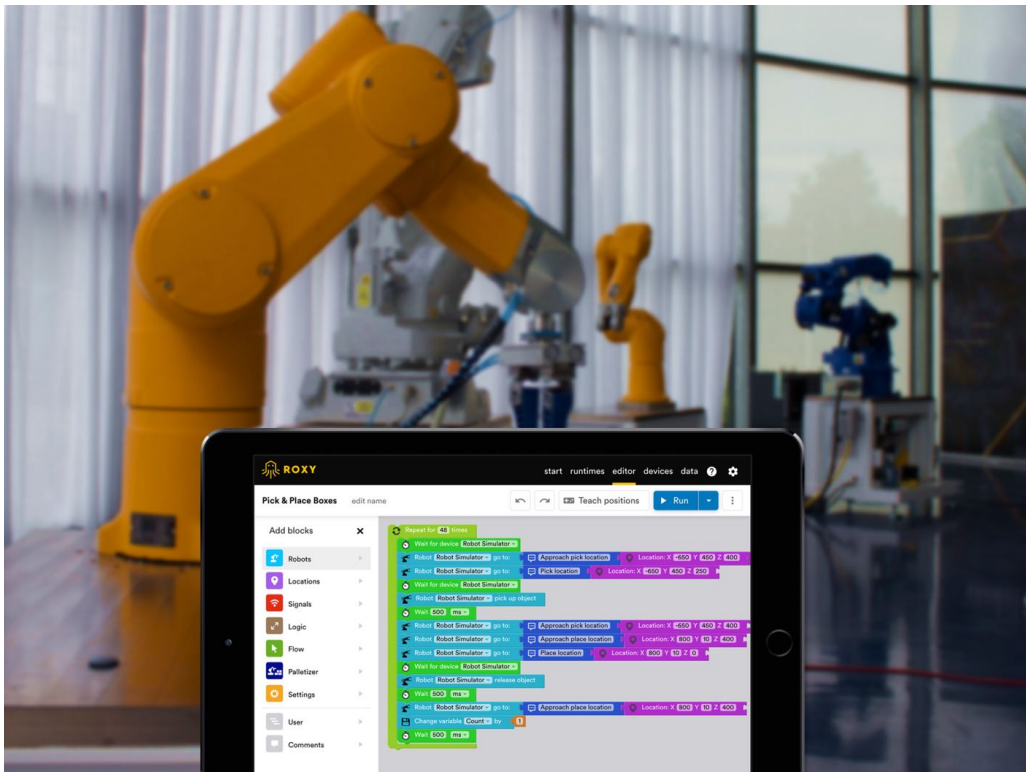
3.3. ábra. Az IRB 1100-as ipari robotkar (forrás: https://new.abb.com/)

3.4. ábra. A YuMi robotkar (forrás: https://new.abb.com/)

A gyakori műveletek, példának okáért a YuMi megfogó vezérlése vagy a felhasználói dokumentáció szintén megtalálható az alkalmazáson belül. Az új robotpozíciók megadása a mozgás blokkokkal, illetve a YuMi robotkar átvezetés funkciójával történik. A blokkok használata mellett ciklusok implementálása is lehetséges. Emellett lehetőség van a kódba hibakezelő blokkok beszúrására, ezáltal a felhasználó egyszerűen képes kezelni a felmerülő problémákat, mint például az ütközést [21].

3.2.2. Roxy Robotics

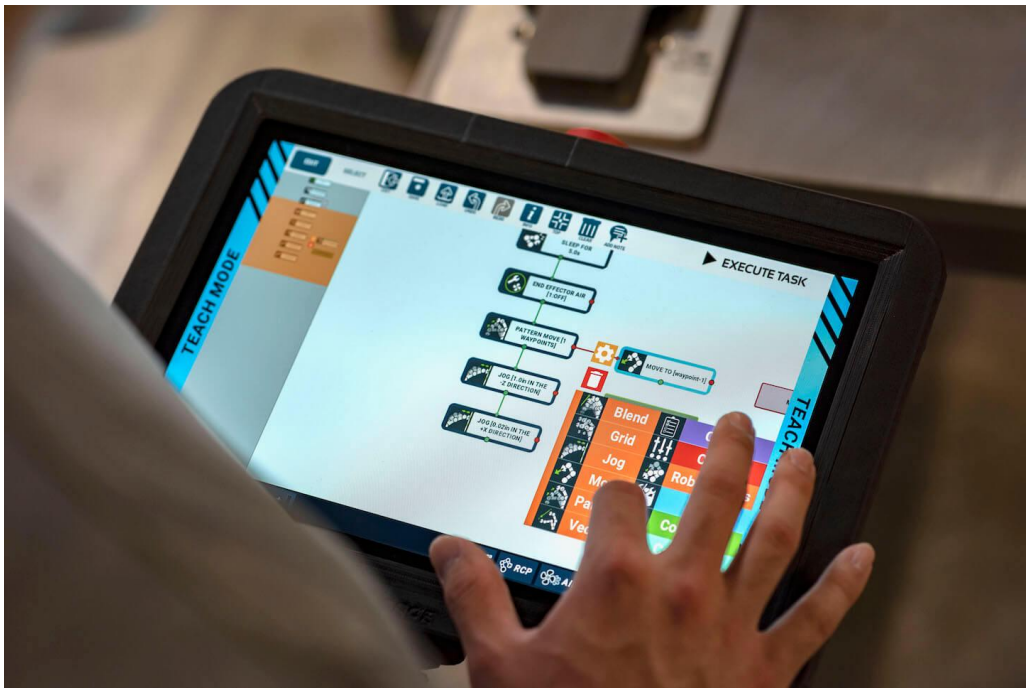
A Roxy Robotics szoftvere hasonlóan a korábbi bemutatott rendszerekhez egy felhasználóbarát, blokk alapú vizuális programozási környezetet biztosít az ipari robotok programozásához. Az univerzális platformnak köszönhetően többféle gyártó robotmegoldásához készíthető programkód, nevezetesen az ABB, FANUC vagy a Stäubli robotkarjaihoz. A szoftver lehetővé teszi a programkód tesztelését is, emellett akár egy virtuális robot segítségével vizualizálni lehet a munkafolyamatot [22].



3.5. ábra. A Roxy Robotics felülete (forrás: https://roxy-robotics.com/en/)

3.2.3. Ready Robotics

A Ready Robotics az elsők között hozott létre egy univerzális platformot, a Forge/OS-t ipari robotok automatizálásához. A Task Canvas szoftver egy felhasználóbarát, folyamatábra alapú, érintőképernyős ”fogd és vidd” módszerrel működik, így robotmárkától függetlenül bárki képes lehet akár egy FANUC, Universal Robots, KUKA, ABB vagy akár egy Yaskawa robotkart a gyártási folyamatoknak megfelelően konfigurálni. A cég nagymértékű testreszabhatóságot és megbízhatóságot kínál az ügyfelek számára, hiszen a robot irányítása mellett az I/O perifériák és szenzorok vezérlésére is lehetőség nyílik a szoftveren belül. Ezáltal akár egy magasfokú precizitást igénylő gyártási folyamat testreszabásához is magabiztosan használható [23].

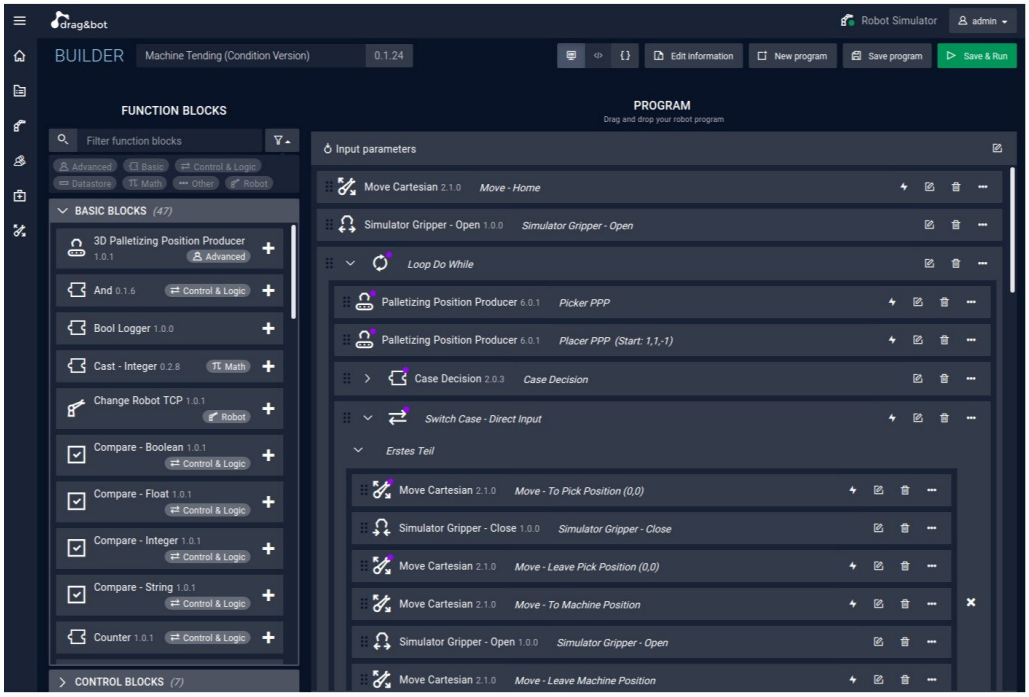


3.6. ábra. A Ready Robotics felülete (forrás: https://www.ready-robotics.com/)

3.2.4. Drag&bot

A Drag&bot cég a robot rendszerek rugalmas programozására robot gyártótól függetlenül kínál megoldást egy felhő alapú szolgáltatással. A grafikus felhasználói felületen 3D-s környezetben lehet a robot cellát és a munkadarabokat vizualizálni, illetve funkcionális blokkokból ”fogd és vidd” módszer segítségével építhető fel a kívánt folyamat [24]. A Builder alkalmazás előre definiált funkcióblokkjai az alap működési metódusokat tartalmazza a gyakran használt robot feladatokra, úgymint:

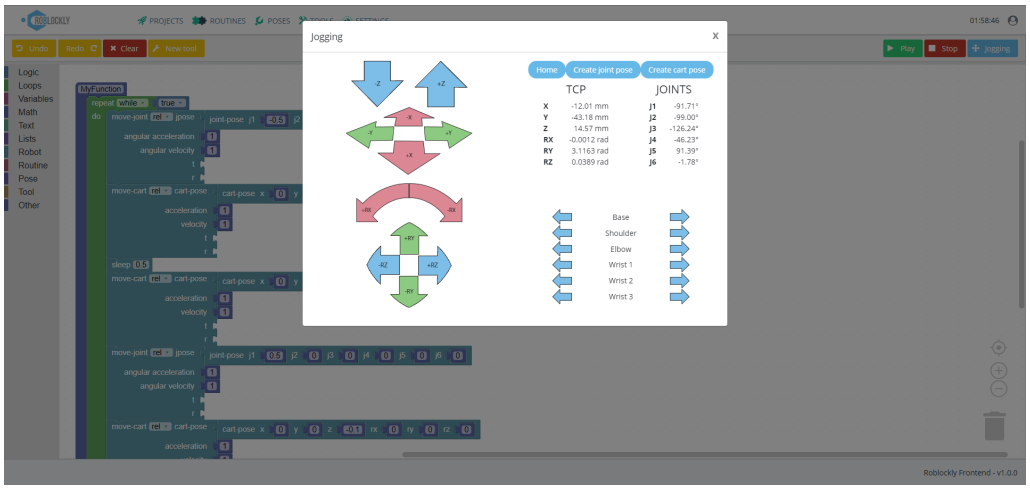
- Robot mozgatás
- Megfogó szerkezet nyitása és zárása
- I/O perifériák olvasása és írása
- Palettázás
- Programozási struktúrák: ciklus, elágazás



3.7. ábra. A drag&bot felülete (forrás: https://dragandbot.join.com/)

4. A Roblockly rendszer architektúrája és technológiai eszközeinek áttekintése

A korábban bemutatásra került rendszereket tekintve beláthatjuk, hogy az ipari robotok programozásának vizuális megközelítésére számos törekvés jelen van a piacon. Míg a Ready Robotics és a Drag&bot egy eltérő megoldásra mutat példát, addig az ABB Wizard Easy Programming és a Roxy Robotics szoftvere felhasználói szempontból hasonló, mint a MaxWhere virtuális térben kialakításra került Roblockly robotprogramozási környezet.



4.1. ábra. A Roblockly webalkalmazás felülete

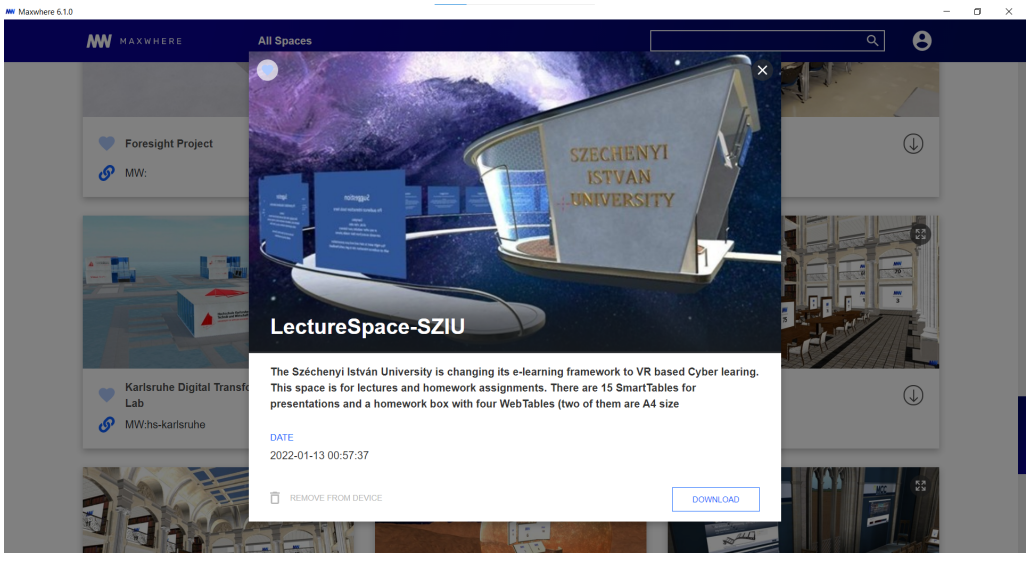
A jelenlegi rendszer egy grafikus kódszerkesztővel rendelkező, web-alapú programozási felület, amelynek segítségével ipari robotok programozását lehet kivitelezni gyártótól és robottípustól független módon. Ezáltal lehetőség nyílik a valós idejű távoli beavatkozásra az ipari robotok vezérlési folyamatai során. A felület és a robot közötti kapcsolatot egy webszerver szolgáltatja, amely egyszerre több klienst is tud futtatni. Azonban egy robot adott időben egyetlen feladatot képes végrehajtani, emiatt egyszerre csak egy felhasználó töltheti fel rá a generált programkódot. A webszerver és a robot közötti kommunikáció többnyire TCP/IP kapcsolattal valósul meg, bár ez a kapcsolat függ a robot típusától és gyártójától is.

A szerver oldalon biztonsági szempontból legfontosabb tényezők, úgymint a felhasználói fiókok kezelése, a hitelesítés és a jogosultságellenőrzés teljesül, illetve az azokhoz tartozó projektek kezelése, létrehozása, törlése is megvalósul. Emellett a

kliens oldalnak biztosítania kell, hogy egy felhasználóbarát kinézettel és működéssel támogassa a rendszer a felhasználótól érkező adatok és interakciók megfelelő kezelését a bejelentkezési és regisztrációs felületen. Továbbá a programkód írására használt oldalon a szerkesztés mellett különböző robot pozíciók felvétele és mentése is megvalósul [25].

A felhasznált technológiák javarészt a JavaScript keretrendszerek közül kerültek felhasználásra, hiszen ezek nyílt forráskódú, szabadon felhasználható rendszereket biztosítanak szerver és kliens oldalon egyaránt [25]. A jellemzően weboldalak fejlesztésére, működésük leírására használt JavaScript programozási nyelv egy objektumorientált, kis erőforrás igényű, prototípus alapú szkriptnyelv, így a futási környezet jellemzően egy webböngésző. Emiatt a kliens oldalon a JavaScript függvények és metódusok a weboldal betöltését követően a szerverrel való kommunikáció nélkül is képesek lefutni [26].

A MaxWhere platform egy különleges digitális élményen keresztül nyújt megoldásokat különböző feladatokra a felhasználók számára az élethű háromdimenziós virtuális környezet által. A szoftver segítségével nem csupán az ipari és kereskedelmi problémákra, de ezen túlmenően az oktatási célú felhasználási módokra is egy újszerű megközelítést alkalmazhatunk a virtuális terek adottságait kihasználva. Ezekre kiváló példa az ipari létesítmények digitális ikerpárjának felépítése, vagy a virtuális események és termékbemutatók megrendezése, illetve a virtuális oktatótermek kialakítása is [27].



4.2. ábra. A MaxWhere környezet (forrás: MaxWhere szoftver)

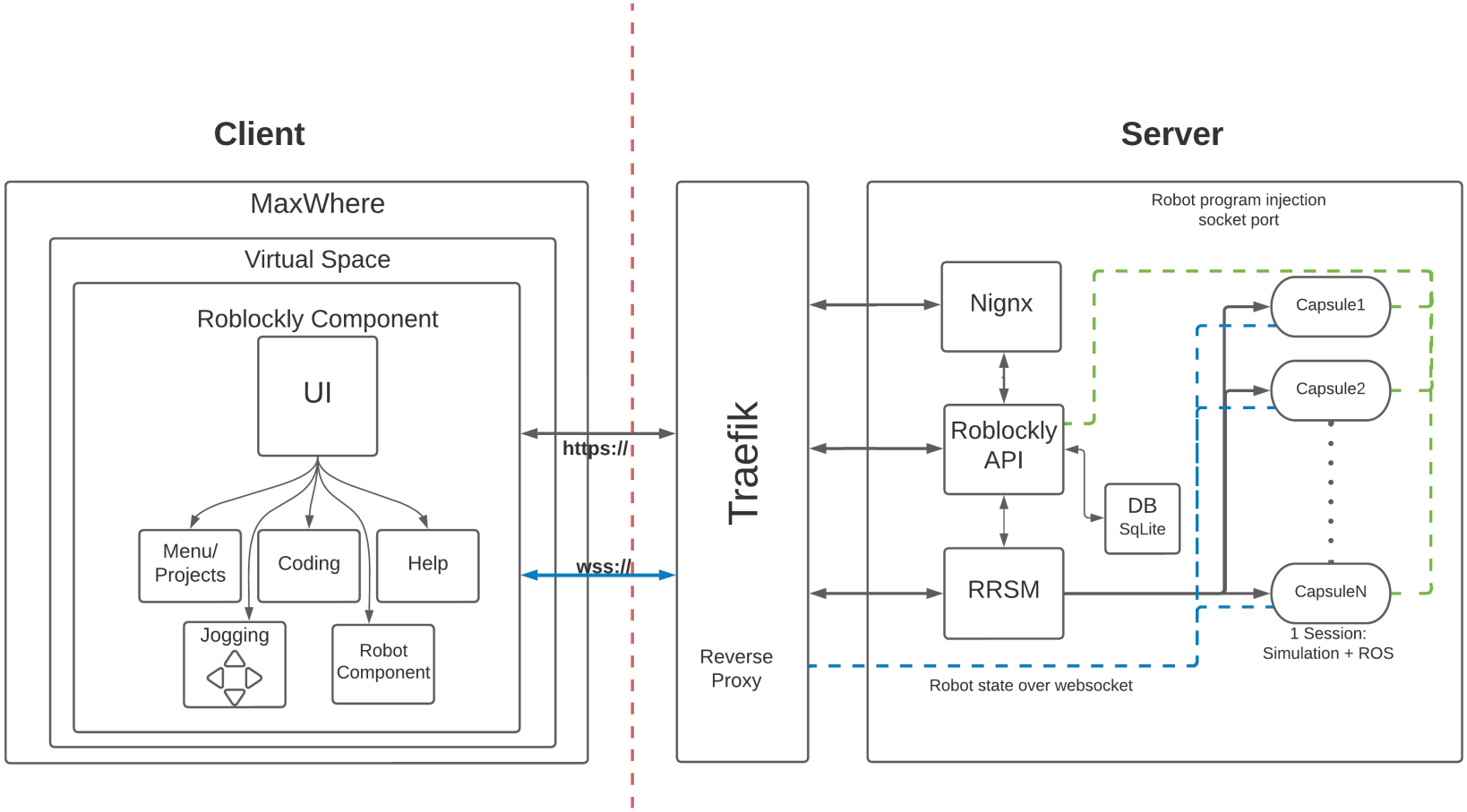
A Roblockly és a Maxwhere kapcsolata egy olyan virtuális térben teljesedik ki, amelyben egy ipari robot digitális példányának mozgatásán keresztül figyelhetjük meg az általunk létrehozott robotprogram működését. A virtuális térben jelenik meg a rendszer kliens oldali része is, vagyis a felhasználói felület a 3D-s alkalmazás részét képezi.

4.1. A meglévő rendszer architektúrája

A projekt architektúráját a 4.3. ábra szemlélteti. A kliens oldalon a MaxWhere virtuális környezet felel a szoftver megjelenítéséért, amely egy „where”-ben, azaz virtuális térben vizualizálja a Roblockly alkalmazást. A korábbi verzióban a MaxWhere a SmartBoard nevű komponensének segítségével – mivel az képes webböngészőként is működni – elérhetővé tette a fejlesztett alkalmazás webszerverén keresztül a grafikus programozói felületet. A bejelentkezést követően az erőforrás menedzser elindította a szimulációs kapszulát a tényleges robotkar reprezentálása érdekében. A virtuális térben megjelenő robotkar a skálázó rendszerrel kommunikált a megfelelő vizualizáció érdekében [25]. A projekt ezen felén dolgozó Peller Gábor feladata a SmartBoard megoldás kiváltása egy letisztultabb és látványosabb felhasználói interfésszel, amely közvetlenül a virtuális térben jelenik meg. Ebben találhatóak meg a különböző menüpontok, ahonnan a projekteket menedzselhetjük, illetve a kódszerkesztő mellett a robotkomponens és annak mozgatására használt gombok is.

Ahogy a rendszer felépítéséből is megfigyelhető, alapvetően a Roblockly alkalmazás több mikroszervizből áll. A Roblockly API, az NginX webszerver, a Roblockly Resource Scaling Manager (RRSM), illetve a Traefik egy-egy különálló Docker konténerben, elszeparáltan futnak, amelyek egymással egy belső hálózat, a RobloclkyNet által állnak összeköttetésben.

Alaposabb egyeztetéseket követően az én munkám elsősorban a szerver oldali rész vizsgálatára irányul, ennek eredményeképp dolgozatom témája a szoftver ezen részének részletesebb ismertetése. Emellett azonban, hogy egy teljes képet alkothassak a rendszer működéséről, írásomban kitérek a kliens oldali technológiák áttekintésére, illetve bemutatásra kerül a rendszer felállításához szükséges Docker keretrendszer is.



4.3. ábra. A Roblockly alkalmazás architektúrája

4.1.1. A Roblockly erőforrás menedzser

Az RRSM, vagyis a Roblockly Resource System Manager a Roblockly kliensek háttérszolgáltatásának biztosítására automatikusan hozza létre és kezeli az úgynevezett Capsule-t, a komponens egész életciklusát követve, ami azt jelenti, hogy a munkamenet lejárását követően a kapszula is megszűnik. A szerviz egy Node.js-ben megírt RESTful protokollon vezérelhető proxy, amely egy, az erőforrások megfelelő elosztásáért felelős skálázórendszer.

A Node.js egy nyílt forráskódú JavaScript-futtatómotoron alapuló szoftverrendszer számos beépített könyvtárral, melyet hálózati architektúrák, valamint skálázható webszerverek üzemeltetésére hoztak létre. A skálázhatóság az adott rendszer vagy folyamat áteresztőképességének növelésére irányul, amelynek jelentősége többek között abban rejlik, hogy a felhasználó előtt rejtve marad a rendszer leterheltsége, és ezáltal egyidejűleg jelentősen több felhasználó kiszolgálására képes [28].

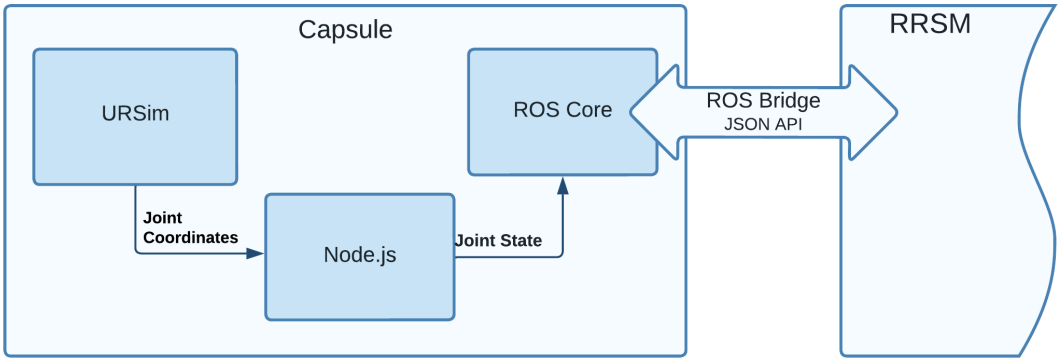
Ez a keretrendszer egy esemény alapú, nem-blokkoló I/O API, ahol többnyire az események aszinkron módon mennek végbe, emiatt a program nem blokkolódik. A legtöbb JavaScript programmal ellentétben nem a webböngészőn keresztül, hanem szerveroldali alkalmazásként fut, implementálja a CommonJS modul egy részét, valamint az interaktív tesztelést is támogatja egy REPL[7] rendszerrel. A keretrendszerben már alapértelmezetten is számos modul található, viszont a Node Package Manager segítségével további, akár több száz könyvtár is telepíthető, valamint saját modulokat is publikálhatunk az npm rendszerbe [28].

A RosLib.js a Robot Web Tools keretein belül került kifejlesztésre, egy standard JavaScript könyvtár, amely a webböngészőből a Robot Operating System-mel történő interakció létrejöttéhez szükséges. A frontend oldalról websocketek segítségével kapcsolódik a ROS Bridge-hez, amely egy JSON API-t biztosít a ROS-funkcionalitáshoz a nem ROS-programok számára. Ezáltal a RosLib képes az alapvető ROS funkciókat ellátni, úgymint a publikálás, feliratkozás, szolgáltatás hívások, actionlib alkalmazása, illetve TF, URDF átalakítás [29].

[7]REPL (Read-Eval-Print-Loop): egy egyszerű interaktív shell programozási környezet, amely a beviteli parancsokat azonnal végrehajtja és az eredményt visszaadja a felhasználónak

4.1.2. A Capsule szimulációs komponens

A szimulációs kapszula egy Universal Robots szimulátor és ROS komponensekből álló, Node.js alapú alkalmazás, amely a robotkar megjelenítéséért és mozgásáért felel. Az Universal Robots ingyenesen elérhető, hivatalos szimulátora az UR robotok teljes funkcionalitásáért felel, míg a Robot Operating System a MaxWhere-ben történő vizualizáció során biztosítja a kartagok térbeli pozíciójának, illetve orientációjának egységes kiszámítását.



4.4. ábra. A kapszula felépítése

A kapszula működésének folyamatát az alábbi pontok határozzák meg:

1. A konténerben futó ROS launch elindít egy ROS Core-t, amely feltölti a ROS paraméterszerverrel a robotleírót, azaz a UR description-t, illetve ezzel egyidőben elindítja a ROS Bridge szolgáltatást és a robot state publisher-t.

2. A Node.js alkalmazás a kapszulában futó UR szimulátortól megkapja a radiánban megadott csuklókoordinátákat, amelyeket ezután bepublikálja egy standard joint state topicba.

3. A robot state publisher a joint state topicba bepublikált koordináták és a robotleírás alapján megalkot egy trasformation topicot, azaz tf-et. Ebben egy időbélyeggel kiegészítve megjelennek az adott kartagok koordinátái Descartes-féle rendszerben megadva, valamint az orientációk is kvaterniókban meghatározva.

4. A ROS Bridge a topic-ok írására és olvasására alkalmas interfész. Tulajdonképpen egy kétirányú átjárót képezve tovább tudja adni nem-ROS alkalmazásoknak, jelen esetben az RRSM komponensnek a ROS üzeneteket, illetve a beérkező JSON fájlokat olvassa és fordítja a ROS Core számára.

4.1.3. A Roblockly API bemutatása

A Roblockly API (Application Programming Interface) a web applikáció szerver oldali része, amely egy reaktív technológiájú Node.js program, Express.js keretrendszerben fejlesztve. A reaktív technológiai megoldás arra utal, hogy a rendszer reszponzivitásra törekszik, amit ellenállóképessége és elaszticitása biztosít, tehát egy hiba bekövetkezése vagy jelentős terhelés esetén is reszponzív marad. Ezek biztosítása érdekében a rendszer aszinkron módon működik.

A Roblockly webszerver a felhasználói fiókok kezelésére fel van készítve, amely azt jelenti, hogy a regisztráció és a projektek kezelése itt történik. A bejelentkezést követően a munkamenet ideje alatt a virtuális robot és a webszerver közötti kommunikáció TCP/IP protokoll kapcsolaton keresztül történik.

Az Express.js egy minimalista, erőforrás-hatékony és rugalmas, JavaScript-ben írt keretrendszer, amely robusztus funkciókat nyújt mobil és webalkalmazások fejlesztéséhez Node.js futtatási környezetben. Ez lényegében a rendszer azon képességére utal, hogy egyaránt képes kezelni a váratlan bemeneteket, illetve a végrehajtás közben előforduló hibákat is. Ezen tulajdonság megléte azt eredményezi, hogy különböző eszközökön is jól használható, a böngészőkkel illetve a kisegítő csomagokkal is kompatibilis. Emellett számos népszerű szerveroldali és full stack keretrendszer alapjául szolgál, független komponenseket biztosít, ezáltal megkönnyíti a kliens oldali programozók munkáját a szerver oldali fejlesztésre való áttérés során [30].

Az adatbázis egy egyszerű SQLite adatázis-kezelő rendszeren alapszik, amely tulajdonképpen a felhasználói adatok tárolásáért felel, amibe beletartoznak az autentikációs információk, valamint az elmentett projektek és programkódok is.

4.1.4. Az NginX szerepe

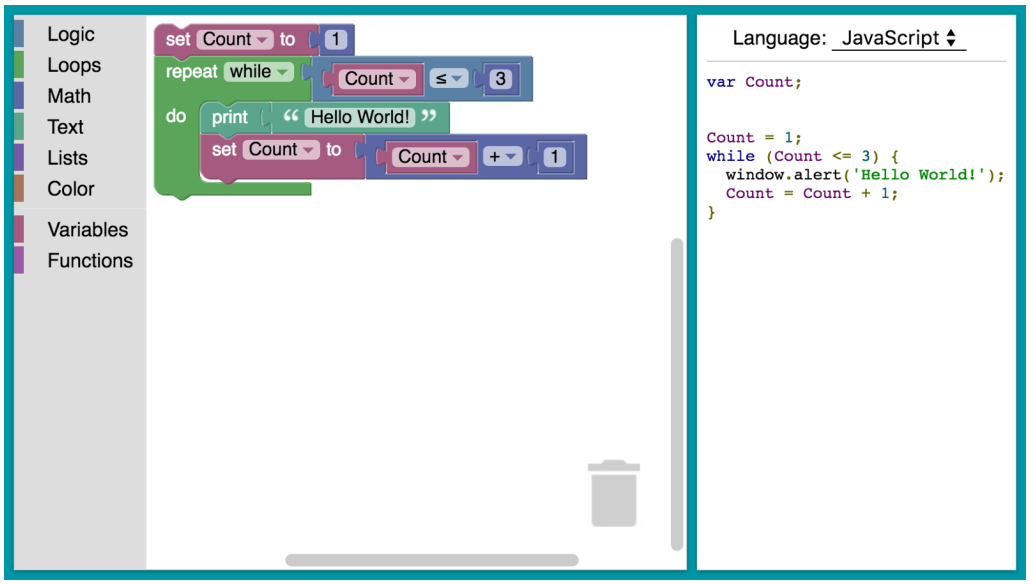
A Roblockly webalkalmazás kliens oldali részéért a Vue.js keretrendszerben megírt komponens felel, melynek alapát egy NginX webszerver képezi. A blokk alapú vizuális programozási környezet itt kerül megjelenítésre.

A Vue.js egy progresszív, JavaScript alapú keretrendszer, amely más, monolitikus keretrendszerrel szemben olyan módon lett megtervezve, hogy fokozatosan lehessen alkalmazni, beépíteni a fejlesztés során. Az alapkönyvtár kizárólag a felhasználói felület nézetére összpontosít, emiatt könnyen integrálható meglévő projektekbe más könyvtárak használata mellett. Többnyire web interfészek és one-page alkalmazások fejlesztése során használják [31].

A felhasznált JavaScript könyvtárak egyike a React.js, amely felhasználói felületek létrehozásához nyújt segítséget. A deklaratív nézetek a felhasználói interfészek tervezése szempontjából hasznos, ugyanis adatváltoztatás esetén kizárólag a megfelelő komponensek lesznek újrarenderelve, ami áttekinthetőbbé teszi a kódot és megkönnyíti a hibakeresést is. A komplex felhasználói interfész létrehozásához komponenseket lehet létrehozni, amelyek logikája JavaScriptben van megírva, így az alkalmazáson belüli adatátvitel sokkal egyszerűbbé és kezelhetőbbe válik [32].

A Bootstrap az egyik legnépszerűbb nyílt forráskódú, CSS alapú, frontend keretrendszer, amit reszponzív mobil- és webfejlesztésre használnak. Többnyire CSS, de adott esetben JavaScript alapú tervező sablonokat biztosít a fejlesztők számára a felhasználói felület kialakítására [33].

A Blockly a Google által fejlesztett, nyílt forráskódú JavaScript könyvtár, amely vizuális programozási szerkesztők felépítéséhez, blokk alapú programozási környezetekhez szükséges nyelvek és eszközök kialakítását segíti elő. Ennek során olyan, egymással összeköthető vizuális elemeket használ, amelyek megkönnyítik a programkód megírását és JavaScript, Lua, Dart, Python, de akár PHP vagy bármilyen más szöveges alapú programkód is generálható belőle. A Blockly könyvtár egyszerű, szintaktikailag helyes kódot generál a szerkesztőben lévő összekapcsolt blokkokból, amely felhasználható különböző alkalmazásokban játékok futtatásához, vagy akár robotvezérléshez is [34].



4.5. ábra. Blockly kódgenerátor (forrás: https://developers.google.com/blockly/)

Ezen könyvtár felhasználásával könnyen létrehozható egy olyan szerkesztő felület, amelyet hozzáadhatunk web-, vagy akár mobilalkalmazásokoz egyaránt. A Blockly egy kliens oldali keretrendszer, emiatt nem igényel támogatást a szervertől, kivéve abban az esetben, ha a rendszert felhő alapú technológiával szeretnénk megvalósítani [34].

4.1.5. Traefik

A Traefik egy nyílt forráskódú, speciális edge router, amely voltaképpen lehetővé teszi a belső hálózat számára, hogy csatlakozhasson a külvilághoz. Számos funkciója mellett automatikusan megtalálja a megfelelő konfigurációs beállításokat a mikroszervizek számára, és az infrastruktúra feltérképezésének köszönhetően képes eldönteni, hogy a beérkező kérést melyik komponensnek kell elküldenie [35].

Jelen esetben a Traefik egy fordított proxy-ként szolgál[8], amely a szerver és a kliens oldali kommunikáció között egy átjárót alkot, vagyis a beérkező és a kimenő https és websocket protokoll kérések ezen keresztül történnek. A kliens és a szerver közötti hálózati forgalom megvalósítása során eltárolja, hogy adott kérésre a backend milyen választ adott, és emiatt később nem küldi el ismételten a szervernek ezt, mivel képes megválaszolni a saját memóriájából.

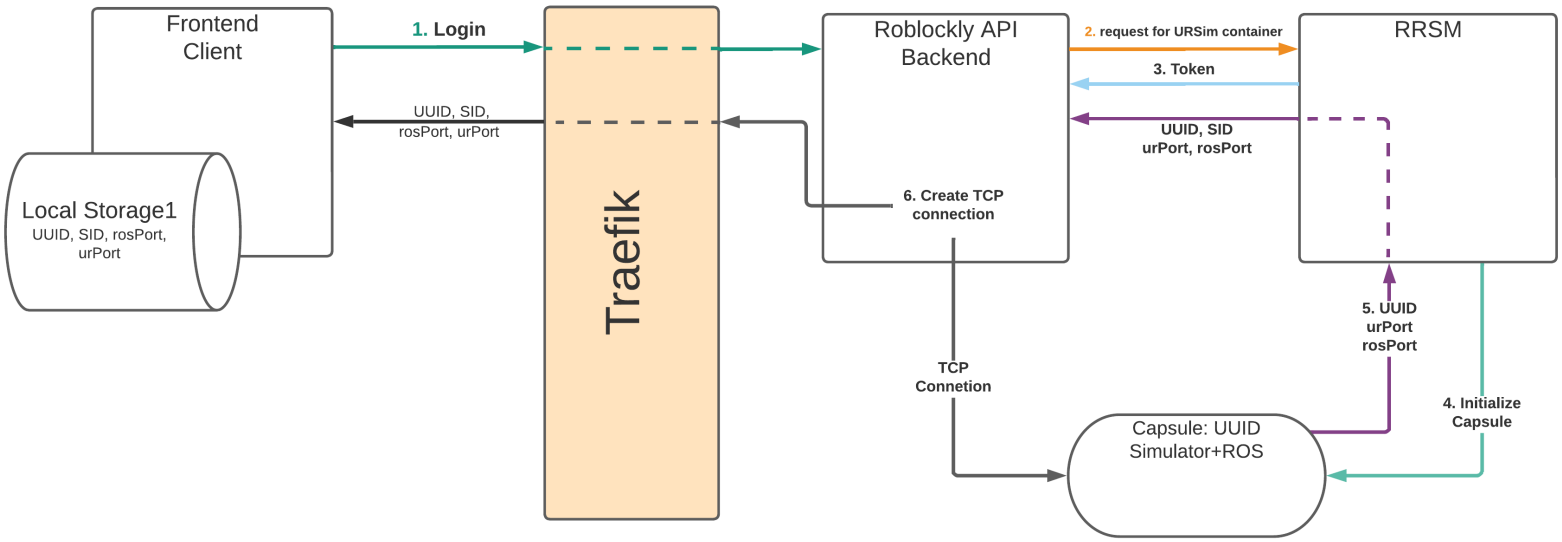
[8]Fordított proxy (reverse proxy): a hagyományos proxyval szemben nem a kliens oldalhoz, hanem a szerverhez van közel, célja elsősorban a szerver terhelésének csökkentése

A szoftver natívan kompatibilis az elterjedtebb cluster technológiákkal, egyebek mellett a Docker és az AWS felhő alapú szolgáltatásaival is, amelyek egyidejű kezelésére képes. Emellett a Traefik használatával nincs szükség külön konfigurációs fájlok karbantartására és szinkronizálására, hiszen minden automatikusan, valós időben történik annak köszönhetően, hogy a szervizekben definiált karakterisztikát képes felismerni. Ezen szempontok számunkra különösen előnyösek a rendszer felépítését, valamint a jövőbeni fejlesztési lehetőségeket egyaránt tekintve [35].

A Traefik indításához a docker-compose.yml fájl létrehozása szükséges, amely a komponens hivatalos Docker image-ét használja. A fájlban megadható számos paraméter, úgymint a portok, a hálózatra vonatkozó parancsok, vagy emellett akár különböző elérési útvonalak és fájlok összekapcsolása is lehetséges a gazdakörnyezet és a konténer között. A Traefik ezenfelül képes log fájlok, vagyis hibanaplók létrehozására is, amely a rendszer működésének monitorozása és a hibák kiszűrése szempontjából előnyös [35].

4.1.6. A komponensek közötti kommunikáció felépítése

Az egyes komponensek közti kommunikációt a 4.6. ábra szemlélteti. A felhasználói felületen a bejelentkezést követően a kliens elküldi a Roblockly API backend részére az autentikációs adatokat, amelyeket továbbít az RRSM számára egy URSim konténer indítási kéréssel. Az RRSM visszaküld egy tokent a backend számára a további kommunikációhoz, és inicializálja a kapszulát egy UUID névvel. A backend a kapszulától visszakapja az RRSM-en keresztül a UUID-t egy session ID névvel, illetve két port számot, az urPort-ot és a rosPortot, amelyen keresztül a robotprogram és a robot pozíciója kerül továbbításra. Ezek az adatok a kliens oldalon lokálisan tárolódnak, ezáltal biztosítjuk, hogy minden kliens más-más kapszula konténerhez férjen hozzá. A backend a kliens oldalról kapott session ID alapján építi fel a TCP kapcsolatot a szimulációs kapszulában futó URsim-mel és küldi tovább a robotkar programkódját a komponensnek.



4.6. ábra. A komponensek közötti kommunikáció

4.2. A Roblockly további technológiai eszközeinek bemutatása

A Roblockly webalkamazás reaktív technológiával került megvalósításra, amely azt jelenti, hogy a rendszer reszponzivitásra törekszik, amit ellenállóképessége és elaszticitása biztosít, tehát egy hiba bekövetkezése, illetve nagy terhelés esetén is reszponzív marad. Ezek biztosítása érdekében a rendszer aszinkron módon működik.

A rendszer alapját felhő és valós idejű web technológiák képezik a MaxWhere interaktív 3D-s környezet mellett. Az egyes technológiák részletesebb bemutatását követően szeretnék kitérni a Roblockly infrastruktúra felállításához szükséges Docker keretrendszerre, valamint említést teszek az AWS felhő alapú platformjáról is.

4.2.1. Docker

A Docker egy igen népszerű, nyílt platform szolgáltatás, amellyel lehetőségünk van alkalmazások fejlesztésére, szállítására vagy akár futtatására is, miközben lehetővé teszi számunkra a mikroszervizek elkülönítését az infrastruktúrától. Ez tehát azt jelenti, hogy a szoftverek teljes életciklusa gyorsabbá válik, hiszen ezen módszerek segítségével jelentősen csökkenthető a kód megírása és a program végleges használata között fellépő idő [36].

Operációs szintű virtualizációval támogatja a konténereknek nevezett szoftver csomagok szolgáltatását. Ezek a konténerek biztonságosan, elkülönítve működnek egymástól, saját könyvtárakat, konfigurációs fájlokat és szoftvereket tartalmaznak, azonban jól definiált csatornákon keresztül képesek az egymás közti kommunikációra. A virtuális gépeknél kevesebb erőforrást használnak, mivel az operációs rendszer kernel szintű szolgáltatásai közös használatban vannak [36].

A Docker a Go programozási nyelven íródott és számos Linux kernel funkciót kihasznál a megfelelő működés biztosítása érdekében. A konténerek kialakításához, azaz annak érdekében, hogy elszigetelt munkaterületekként működhessenek, a Docker az úgynevezett névtér technológiát használja. Egy konténer futása során egy névtérkészletet hoz létre, ami az izolációt biztosítja, mivel a konténerek hozzáférése erre korlátozódik [36].

Architektúráját tekintve egy kliens-szerver kapcsolaton alapszik, ahol a Docker kliens REST API, UNIX socket vagy hálózati interfész segítségével kommunikál a Docker daemon-nal, ami a konténerek fordításáért, futtatásáért és elosztásáért felel [36].

4.2.2. Docker Compose

A Docker Compose tulajdonképpen egy olyan eszköz, amelynek segítségével képesek lehetünk többkonténeres Docker alkalmazások definiálására és futtatására. Egy YAML formátumú konfigurációs fájl segítségével egyszerűen beállíthatóak a környezeti változók, a traefik és tűzfalbeállítások, illetve a docker konténerekre jellemző kezdeti paraméterek is. Példának okáért a konténer neve, TCP/IP portszáma, vagy akár a host IP címe is itt van definiálva, amely a külvilággal való kommunikációhoz szükséges.

A Docker Compose a fejlesztési folyamattól kezdve a tesztelésen át az éles környezet létrehozásáig minden fázisban használható, a tesztelést és a CI (Continuous Integration) munkafolyamatot is jelentősen megkönnyíti. A használata három alaplépésre bontható. Elsőként a Dockerfile-ban definiálni kell a komponens környezetét, amely segítségével könnyedén telepíthető, hordozható lesz a szoftverünk. Ezután a fentebb említett docker-compose.yml fájlban felsorolt szolgáltatások az indítást követően egyszerre tudnak futni egy izolált környezetben, amelyet a docker-compose -up paranccsal hajthatunk végre [37].

Amellett, hogy a Docker Compose az indításhoz szükséges környezet létrehozására is alkalmas, az adott architektúra egész életciklusát képes kezelni, hiszen az újraindítás és leállítás mellett a futó szolgáltatások állapotát és naplózási fájljait is megtekinthetjük, illetve a különböző komponenseken parancsokat is futtathatunk a segítségével [37].

A teljesség igénye nélkül az alábbi felsorolásban találhatóak a fontosabb Docker parancsok:

- docker build [dockerfile] -t [filepath]: a megadott Dockerfile alapján létrehozza a Docker image-t
- docker ps -a: listázza az összes konténert
- docker images: listázza az összes képfájlt

- docker stop [name/ID]: megállítja az adott konténer futását
- docker rm [name/ID]: törli az adott konténert (kizárólag a leállítás után lehetséges)
- docker rmi [name]: törli az adott képfájlt (kizárólag akkor, ha nincs létrehozva belőle futó konténer)
- docker exec -it [name/ID] bash [command]: az adott parancsot a konténeren hajtja végre
- docker inspect [name]: megadja a konténerről elérhető összes információt
- docker logs: amennyiben definiálva van a YAML fájlban, a megadott elérési útvonalon lévő naplófájlt megtekinthetjük
- docker images -f dangling=true: kilistázza a tesztelés és fejlesztés során létrejött, címke (tag) nélküli képfájlokat
- docker rmi $(docker images -f dangling=true -q): biztonságosan törli az összes címke nélküli képfájlt
- docker system prune -a: törli azokat a képfájlokat is, amelyekre nincs konténerhivatkozás. Az -a opció használata olyan képfájlok törléséhez is vezethet, amelyeket meg szeretnénk tartani egy esetleges korábbi verzióra való visszaállás miatt, emiatt használata körültekintést igényel.

A Roblockly infrastruktúra alapját a négy főkomponens, a Traefik, az RRSM, a Roblockly API és az NginX előre definiált Docker képfájljaiból készített konténerek alkotják, amelyeket egy belső hálózat, a RoblocklyNet köt össze. A docker-compose.yml fájlban megadható a képfájl neve, a létrehozandó konténerek elnevezése, a portok és környezeti változók definiálása is. Az alábbi kódrészlet a fájl NginX-re vonatkozó része:

```
nginx:
  image: nginx:latest
  container_name: nginx
  environment:
    - URL=http://roblocklytest.maxwhere.com
    - REACT_APP_ENV=production
  volumes:
    - ~/roblockly-compose/nginx/default.conf:/etc/nginx/conf.d/default.conf
  networks:
    - RoblocklyNet
  expose:
    - "80"
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.frontend-secured.entrypoints=web"
    - "traefik.http.routers.frontend-secured.rule=Host(`roblocklytest.maxwhere.com`)"
```

4.7. ábra. NginX definiálása a docker-compose.yml fájlban

4.2.3. AWS

Az Amazon Web Services egy felhő alapú platform, amely különböző szolgáltatásokat foglal magában, nagyon nagy számítási kapacitás biztosításával [38]. Az alábbi felhasznált szolgáltatások segítségével biztosítani lehet a meglévő rendszer skálázhatóságát is:

- Route53: DNS web szolgáltatás
- EC2 (Amazon Elastic Compute Cloud): virtuális gépek bérlésére irányuló szolgáltatás
- Elastic Load Balancing: terheléses menedzsmentre irányuló szolgáltatás
- Code Build: folytonos integrációs szolgáltatás a forráskód lefordítására, tesztek futtatására és a telepítésre kész szoftvercsomagok elkészítésére

A felhasznált főbb technológiai megoldások alkalmassá teszik a rendszer komponenseit a környezet egy felhő alapú szolgáltatásban való kiépítésére. Abból adódóan, hogy a MaxWhere szoftver egy AWS környezetben működik, kézenfekvő megoldás lehet a Roblockly üzemeltetését is ezen a platformon megvalósítani.

5. A lokális teszt környezet kiépítése

Első lépésként egy virtuális gépen hoztam létre egy Linux környezetet, ahol a Docker és a Docker Compose telepítését követően alaposabban is megismerhettem a rendszer működését. Mivel a meglévő éles környezet más konfigurálással és hálózati beállításokkal rendelkezik, el kellett érnem, hogy a Roblockly rendszer lokálisan működjön a saját gépemen. Az implementáció részletes leírását a Mellékletek fejezet 10.1. alfejezetében ismertetem.

5.1. Felmerült problémák és megoldásaik

A kihívást elsősorban az jelentette, hogy a projekten sokan dolgoztak már korábban, a hozzá tartozó fejlesztői dokumentáció emiatt gyakran félrevezető volt, mivel nem minden esetben voltak pontosan megjelölve a rendszer aktuális állapotára vonatkozó információk, vagy gyakran hiányoztak, különös tekintettel a fejlesztői teszt környezet kialakításával kapcsolatban.

A Dockerfájlok felépítését tekintve elavult megoldásokat tartalmaztak, amelyek nem követték a „best practice”, vagyis a bevált gyakorlati elveket. A Docker image-ek lényegében úgy épülnek fel, hogy minden egyes lépés egy új réteget húz az előzőre, ezért törekedni kell a tömör, egyszerű felépítésre, ami által a képfájl buildelésének ideje jelentősen csökkenthető. A folyamat gyorsítására szolgálhat még az a megoldás is, hogy a ritkán vagy egyáltalán nem módosuló szinteket a fájlban korábban definiáljuk, mivel a Docker az újrabuildelés során egy cache memóriából állítja vissza azokat, amelyekben nem történt változás a megelőző verzióhoz képest.

A buildelést követően eleinte hibára futott a program, aminek sok esetben az oka a fájlban definiált alkalmazásrétegek verziója volt, hiszen már nem volt támogatott az operációs rendszer, vagy a Docker újabb verziója által. Emiatt ezeket felül kellett vizsgálni és javítani a következő próbálkozás előtt.

Számos esetben találkoztam azzal a megoldással, hogy a host rendszeren egy adott felhasználó könyvtárában tárolt (többnyire konfigurációs) fájl lett átmásolva a képfájlba, hogy az adott komponens az indítást követően az abban megadott értékeket vegye figyelembe. Könnyen belátható, hogy ez a megvalósítás hosszú távon

a rendszer sérülékenységéhez vezethet, valamint ellehetetleníti a Docker környezet azon tulajdonságának kihasználását, amely az alkalmazás hordozhatóságát és egyszerű telepíthetőségét biztosítja alapesetben. Erre példa az RRSM Dockerfájljának buildelése során kapott alábbi hibaüzenet is.

```
Cloning into 'roblockly'...
Removing intermediate container fc250977b039
 ---> a5b89824d924
Step 4/10 : COPY config.json /roblockly-rsm/config
 ---> d1d07c4d0490
Step 5/10 : WORKDIR /roblockly-rsm
 ---> Running in 9c831aa2e3f1
Removing intermediate container 9c831aa2e3f1
 ---> afac205cc961
Step 6/10 : RUN npm install
 ---> Running in 7459192bb539
npm WARN saveError ENOENT: no such file or directory, open '/roblockly-rsm/package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open '/roblockly-rsm/package.json'
npm WARN roblockly-rsm No description
npm WARN roblockly-rsm No repository field.
npm WARN roblockly-rsm No README data
npm WARN roblockly-rsm No license field.

up to date in 0.512s
found 0 vulnerabilities

Removing intermediate container 7459192bb539
 ---> b0258eefc42e
```

5.1. ábra. Hiba az RRSM image buildelése során

Továbbá biztonsági rést jelent a Dockerfájlban a privát repository-kból való klónozás is, mivel ez azt hozza magával, hogy meg kell adnunk a verziókövetőhöz a saját SSH kulcsunkat is a fájlon belül. Emiatt bárki, aki hozzáfér az image-hez, a verziókövetés miatt könnyedén visszafejtheti az adott rétegből a hitelesítési adatainkat, amivel jelentős károkat okozhatnak egy éles rendszerben. Ezen problémára megoldást jelenthet az is, ha a tokeneket egy másik fájlban tároljuk, és környezeti változóként adjuk át a Dockerfájlnak, vagy az image-ket egy felhő alapú registryben tároljuk, úgy, mint a DockerHub vagy a GitLab saját registry-je.

A teszt környezet felállításához nem volt elegendő a konfigurációs fájlokat beállítani, amire abból lehetett következtetni, hogy a ROS kapszula nem a saját virtuális gépemen jelent meg a Docker konténerek között, hanem az éles szerveren indult el a teszt szerver kliensén való bejelentkezést követően. Félrevezető volt, hogy habár a dokumentáció szerint, környezeti változóként meg lett adva az RRSM komponensnek a host IP címe, illetve a docker-compose.yml fájlban is helyesen volt konfigurálva a Traefik számára a hálózati beállítás, mégsem az elvárt működés szerint viselkedett a rendszer. A web

applikáció és az RRSM komponens alaposabb vizsgálata során kiderült számomra, hogy az éles szerver domain-je be van ágyazva a kódba több helyen is a session létrehozására szolgáló folyamat során. Ennek javítását követően a webalkalmazás már a virtuális szerver RRSM-jére próbált kéréseket küldeni, ennek köszönhetően már nem kommunikált az éles szerverrel és nem jött létre ott tévesen kapszula, azonban a kéréseket nem tudta továbbítani a rendszer az RRSM számára, ugyanis a virtuális gépen a Docker szerviz alapértelmezett beállításai blokkolták ezeket.

A szolgáltatások meghívásának tesztelését az Insomnia REST API klienssel végeztem, melynek segítségével a konténerek által használt http kéréseket küldtem a virtuális gép számára. Emellett a Linux szerveren a curl parancs segítségével próbáltam reprodukálni a kéréseket az egyes komponensek felé, amelyekre adott válaszból igyekeztem feltárni a hiba forrását.

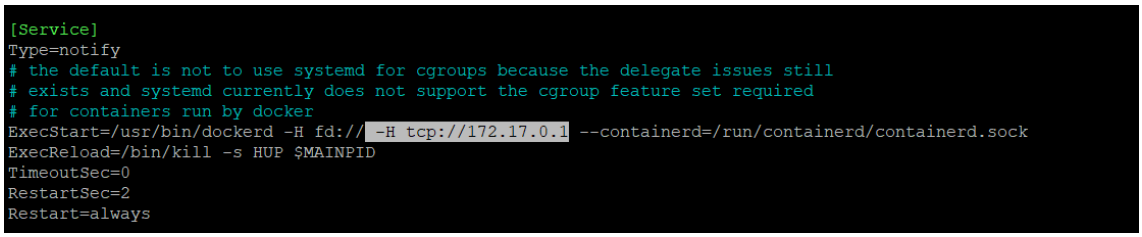
A probléma megoldásához a host rendszer portját elérhetővé kellett tenni a Docker számára. Erre több lehetőség is létezik, az egyik, hogy a localhost IP címét, vagyis a 127.0.0.1-et adjuk meg, amennyiben egy korábbi verziót használunk. A Linux natív Docker 20.10-es verziója óta azonban a host.docker.internal funkción keresztül is megadható a Docker bridge interfész, vagyis a docker0 IP címe, a 172.17.0.1, amely jelen esetben a megfelelő módszer a rendszer tulajdonságait tekintve. Fontos azonban rávilágítani egy ezekhez hasonló, ám nagymértékben kártékony módszerre is, amikor a host 0.0.0.0 portját tesszük elérhetővé, ugyanis ilyen esetben a host minden csatornán hallgatózik és bármilyen beérkező kérést elfogad az internetről. Habár a kívánt eredményre jutunk ezzel is, és a konténerek között létrejön a kapcsolat, komoly biztonsági rést hagyhatunk a rendszerünkön ezáltal.

A fentebb leírt hiba javításához a Docker service, az RRSM config.json és a docker-compose.yml fájl RRSM-re vonatkozó részének módosítására is szükség volt, ezen változtatások lépéseinek folyamatát az alábbi pontok szemléltetik.

1. Docker config módosítása:

   A docker0 (vagy ha létezik, akkor a docker1) hálózati interfész IP címének megadása a Docker serviceben:

   - sudo nano /lib/systemd/system/docker.service



5.2. ábra. A Docker Service fájl

- sudo systemctl daemon-reload
- sudo systemctl restart docker

2. RRSM config.json módosítása:

   ”host”: ”http://172.17.0.1”,

   ”ip”: ”172.17.0.1”,

   ”port”: 2375

3. A docker-compose.yml-ben az RRSM konténerre vonatkózó rész kiegészítése az extrahosts paraméterrel, ahol megadjuk a ”host.docker.internal:host-gateway” változót:

```
rrsm:
  #build: ~/RRSM
  image: rrsm:latest
  container_name: rrsm
  expose:
    - "3000"
  networks:
    - RoblocklyNet
  labels:
    - "traefik.enable=true"
  extra_hosts:
    - "host.docker.internal:host-gateway"
```

5.3. ábra. A docker-compose.yml fájl módosítása

A kezdeti nehézségeken túllendülve a Roblockly rendszer elindult a virtuális Linux szerveren, amely azt jelentette, hogy a MISTEMS által szolgáltatott fizikai szerveren is megkezdődhetett a környezet és az infrastruktúra kiépítése a szolgáltatások számára. A teszt környezet kialakításának során fény derült a rendszer bizonyos sérülékenységeire, ezért különös gondossággal kellett eljárni a Dockerfájlok felépítése, illetve a hálózati beállítások konfigurálása közben is a fizikai szerveren történő megvalósítás alkalmával.

Emellett, mivel a manuális image buildelés akár több tízperces procedúra volt Dockerfájlonként, a képfájlokban a rétegek csökkentése (és ezzel párhuzamosan a build idő csökkentése) a továbbhaladás szempontjából jelentős volt. Továbbá a képfájlok átkerültek a projekt GitLab Registry-jébe, amely egyrészről biztonságosabb tárolási módja a legutolsó stabil verziónak, valamint a manuális buildek kiválthatóak egy egyszerű Docker pull paranccsal is. Mindezek mellett azonban a fejlesztés menetének megkönnyítése céljából igény keletkezett egy CI/CD pipeline kiépítésére is a rendszer mellé.

6. A CI/CD pipeline megvalósítása

A CI/CD folyamat egy olyan agilis szoftverfejlesztési módszertan, amelyben a DevOps[9] fejlesztők rendszeresen, akár naponta többször is egy közös verziókövető felületet használva integrálják a kódban végzett módosításaikat anélkül, hogy a rendszer adminját vagy a DevOps csapatot kellene megkérniük ennek végrehajtásához. A CI/CD automatizálja a fejlesztés és telepítés lépéseit, vagyis a kód futtatását, tesztelést és a szoftver új verziójának biztonságos üzembe helyezését is. Ezáltal egy sokkal hatékonyabb munkafolyamat kerül kialakításra, hiszen a kisebb változtatások is gyorsan és egyszerűen elérhetővé válnak az egész csapat számára. Ennek eredményeképp minden új kódrészlet ellenőrzésre kerül, amely lehetővé teszi a problémák korai detektálását és azonnali javítását, így minimalizálva a javítandó hibák mennyiségét. A fejlesztés során jelentős időt lehet nyerni a módszer implementálásának köszönhetően, valamint egy skálázható folyamatot lehet kiépíteni ezáltal [39].

[9]DevOps: a development (Dev, vagyis fejlesztés) és az operations (Ops, vagyis üzemeltetés) szavakból álló mozaikszó, amely az így kialakult csapatra, valamint a folyamat során az általuk használt technológia egyesítésére utal

A CI/CD pipeline működése tulajdonképpen megegyezik a gazdarendszeren futtatott CLI[10] parancsokkal. A folyamathoz szükséges egy Git repository-ban tárolt kódbázis, illetve egy .gitlab-ci.yml fájl a program gyökérkönyvtárában, amely a konfigurációkat, szkripteket és parancsokat tartalmazza, amelyeket futtatni szeretnénk. A szkriptek különböző feladatokban, azaz „job”-okban vannak megírva, amelyek végrehajtási sorrendjét a különböző szintek, vagyis „stage”-ek határozzák meg [40].

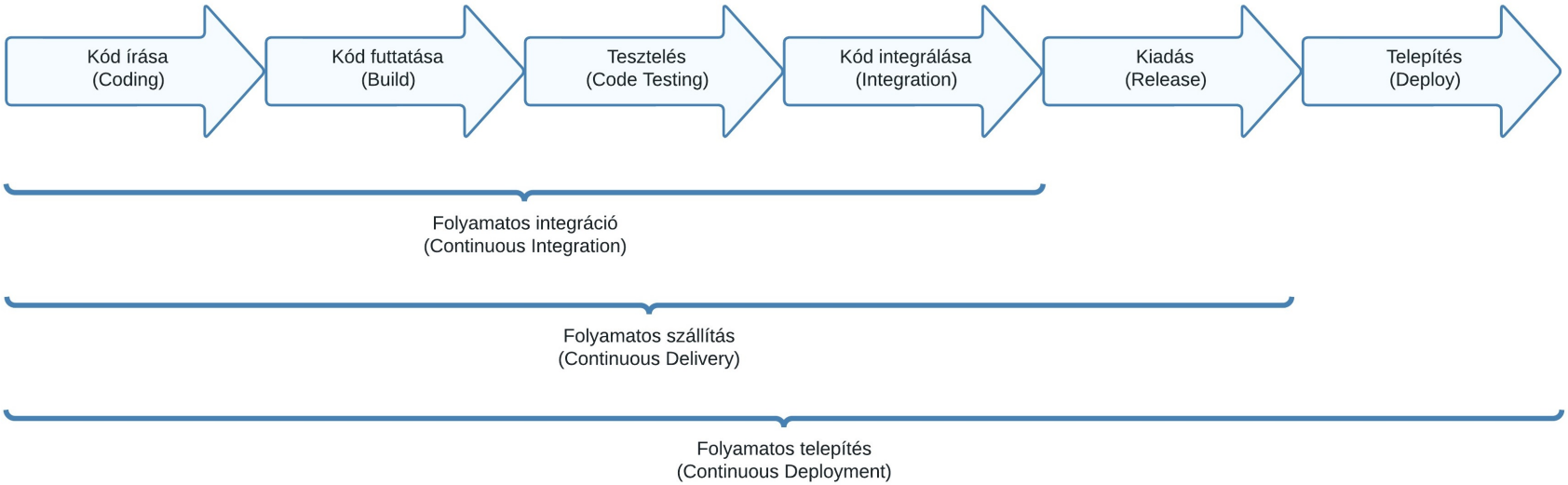
A módszertan három, egymásra épülő szakaszt foglal magában:

Continuous Integration (CI): A folyamatos integráció a fejlesztés és tesztelés mindennapos eljárását könnyíti meg az automatikus buildelés és tesztek futtatása által.

Continuous Delivery (CD): A folyamatos szállítás során a fejlesztői csapat egy rövid cikluson belül képes a szoftvertermék fejlesztésére, amely biztosítja, hogy az alkalmazás újabb verziója egyszerűen átadható legyen az ügyfelek részére. Ennek utolsó lépése manuális beavatkozást igényel.

Continuous Deployment (CD): A folyamatos telepítés vagy üzembe helyezés lényegében az előző lépés automatizálása. Ez segít a tesztelőknek ellenőrizni a kódbázis helyességét és a verzió stabilitását.

[10]CLI (Command Line Interface): Parancssoros felhasználói felület



6.1. ábra. A CI/CD folyamatok

6.1. A fejlesztés rendszerterve

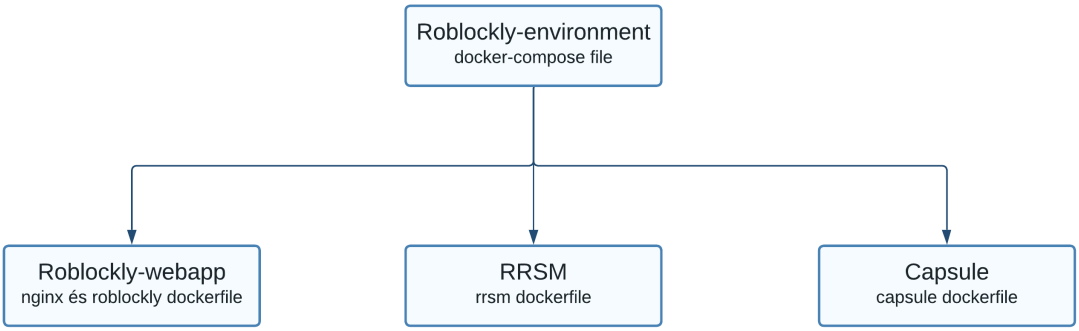
Számos CI/CD eszköz van jelen a piacon, ilyen például a Jenkins, CircleCi, vagy a GitLab beépített CI/CD API-ja is. Munkám során a GitLab ingyenes funkciójának használatát találtam kézenfekvő megoldásnak, ugyanis ez jelentősen egyszerűbbé teszi a projekt menedzselését, hiszen nem kell egy harmadik fél által nyújtott szolgáltatást igénybe venni.

Az egyeztetéseket követően megállapodtunk, hogy a megvalósítás folyamán alapvetően a Roblockly alkalmazáshoz a testing és a production architektúra kialakítására van szükség elsősorban, hiszen a fejlesztés jelenlegi állapotában a staging környezet még nem lenne kihasználva. Ugyanakkor a szoftver üzemeltetésének és kiadáskezelésének[11] előrehaladása nagy valószínűséggel meg fogja követleni a jövőben a staging környezet meglétét is. Ez azt a célt szolgálja, hogy a production környezetbe való szállítás előtt kiszűrésre kerüljenek az esetlegesen fennálló hibák még a staging környezetben.

6.2. A multi-projekt pipeline implementálása

A GitLab CI/CD folyamatot kiterjeszthetjük több különböző projektre is, ezáltal egy adott projektben lévő pipeline kiválthatja más projektben szereplő pipeline indítását is. Ehhez a .gitlab-ci.yml fájlban egy úgynevezett trigger job létrehozására van szükség [41]. Az egyes pipeline-ok indításához egy rendszerszintű repository-ban lévő vezénylő vagy szervező folyamat (orchestrator pipeline) definiálása szükséges [41]. Ez az adattár a docker-compose.yml fájl mintáját tartalmazza, illetve az alsóbb szinteken lévő CI/CD folyamatok indítását ütemezi, majd a teljes rendszert újraindítja a docker-compose down és docker-compose up -d parancsok segítségével. A multi-projekt pipeline architektúráját a 6.2. ábra szemlélteti.

[11] Release Management: Kiadáskezelés

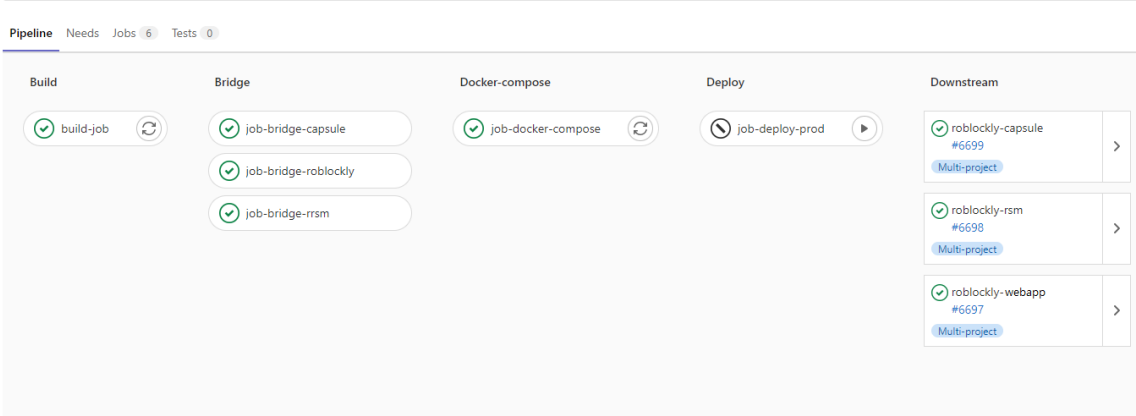


6.2. ábra. A multi-projekt pipeline kialakítása

Az adott rendszerre vonatkozó értékekkel és változókkal példányosított docker-compose.yml fájlt a teszt szerveren az /opt/roblockly-compose elérési útvonalon lévő közös mappába helyeztem el, ezáltal - a docker csoporthoz tartozó - felhasználók akár manuálisan is újra tudják indítani a rendszert, ugyanakkor a GitLab Runner is könnyedén eléri a folyamat futása közben.

A vezénylő pipeline sajátossága, hogy a .gitlab-ci.yml fájljában definiált feladatok között megtalálhatóak az úgynevezett bridge feladatok, amely elnevezési konvenció indikálja, hogy egy triggert tartalmaz a leáramló (downstream) CI/CD pipeline indításához. Egy adott job kizárólag egyetlen szkript vagy trigger futtatásért felelhet, a kettő együtt nem szerepelhet a leírásban.

Első lépésben a jelenlegi fő projektekhez - vagyis az RRSM, a Roblockly és a Capsule forráskódjához - egyesével konfiguráltam fel a hozzájuk tartozó CI/CD folyamatokat. A művelet során létrehoztam és regisztráltam a szükséges GitLab Runnereket a teszt szerveren, beállítottam a végrehajtó típusát, valamint megadtam a pipeline lefutásához a szükséges .gitlab-ci.yml fájlokat a projektek gyökérkönyvtárában. Az így létrehozott pipeline-ok ezáltal az egyes változtatások hatására lefutnak és újragenerálják a teszt szerveren lévő docker image-ket. Amennyiben a folyamatok indítása a rendszerszintű, roblockly-compose adattárból történik, a három pipeline lefut, és a teljes rendszer a legfrissebb képfájlokból létrejött konténerekkel kerül újraindításra.



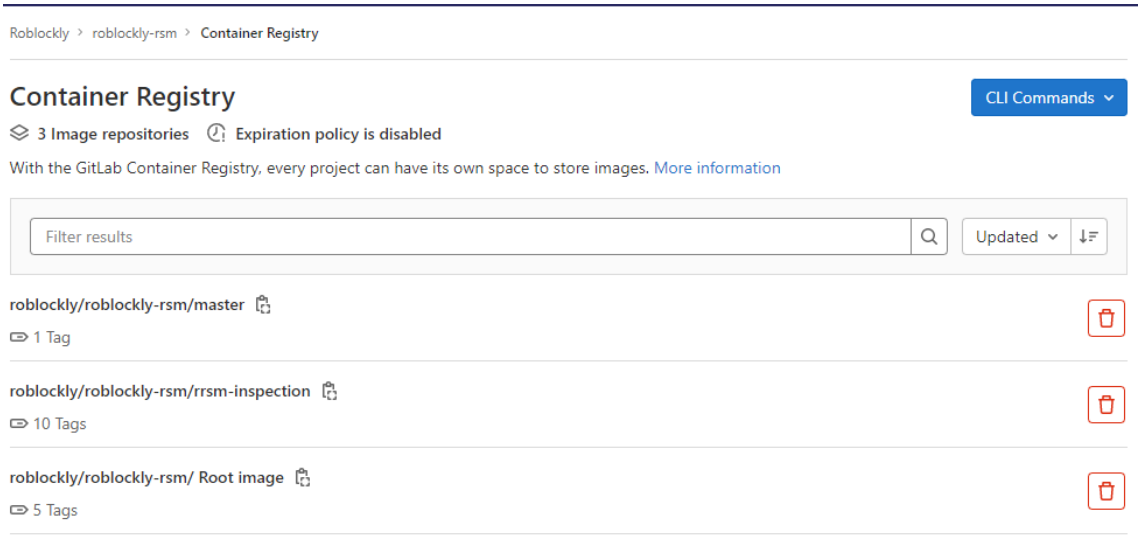
6.3. ábra. A multi-projekt pipeline

6.3. Az implementáció során felhasznált technológiai eszközök

A CI/CD pipeline kiépítéséhez maintainer szerepkör megléte szükséges az adott GitLab adattárhoz, ugyanis ekkor válnak elérhetővé a beállításhoz szükséges funkciók. Fontos megemlíteni még, hogy az egyes branch-eken, azaz ágakon ugyanannak a .gitlab-ci.yml fájlnak kell szerepelnie a megfelelő működés érdekében.

6.3.1. A GitLab Registry

A GitLab Registry a Docker képfájlok tárolására biztosított projekt vagy csoport szintű tárhely, amely egyszerű megoldást nyújt a különböző verziók használatára és karbantartására is. Munkám során a projekt szintű tárhely használatára esett a választásom, mivel az egyes projektek a rendszer egy-egy komponensének feleltethetőek meg, amelyekből a konténerek képfájljai épülnek fel [42]. Az adott projekthez tartozó CI/CD folyamat úgy került felkonfigurálásra, hogy a docker képfájl buildelését követően a GitLab Registry-be is feltölti a megfelelő címkével ellátott képfájlt, ezáltal az egyes verziók könnyedén visszakereshetőek lesznek.

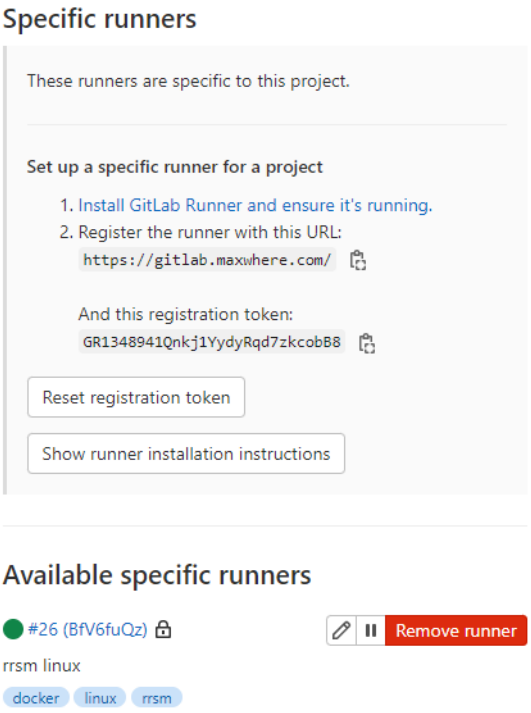


6.4. ábra. A GitLab konténer tárhelye

6.3.2. A GitLab Runner

A GitLab Runner a GitLab CI/CD-hez tartozó alkalmazás, amely a pipeline-ban szereplő feladatok végrehajtásáért felel. Nyílt forráskódú, Go nyelvben írt program, emiatt semmilyen nyelv specifikus követelményt sem kell teljesíteni a különböző operációs rendszereknek és eszközöknek, abban az esetben, ha képesek a Go alapú bináris fájlt futtatni. Az alkalmazás telepítését követően egyedi runner-eket szükséges regisztrálni, amelyek a CI/CD feladatok futtatásáért felelős példányok. A regisztrációt követően tulajdonképpen egy kommunikációs csatorna épül fel a GitLab példány és a számítógép között, amelyen a GitLab Runner telepítve van. Alapesetben a runnerek ezen a gépen hajtják végre a feladatokat, azonban konténerekben, Kubernetes clusterben vagy automatikusan skálázott felhő rendszerekben is működhetnek [43].

A Gitlab Runner regisztrálása az a folyamat, amely összeköti a runnert egy vagy több GitLab példánnyal. Több runner regisztrálása is lehetséges ugyanazon a gazdarendszeren különböző beállításokkal [44]. A runner típusát tekintve lehet megosztott, csoport vagy projekt-specifikus is. Munkám során arra jutottam, hogy a Roblockly szoftverhez az utolsó típusú runner alkalmazása lenne a legmegfelelőbb, hiszen az egyes projektek a különböző docker konténerek forráskódját tartalmazzák. Emiatt külön regisztrálásra került az RRSM, a Roblockly és a Capsule projekt runnerje is a szerverre.



6.5. ábra. Az RRSM projekthez tartozó specifikus GitLab Runner

A telepítés és a regisztrálás dokumentációja a Mellékletek fejezet 10.2. alfejezetében kerül részletezésre.

6.3.3. A GitLab Executor

A GitLab Executor segítségével saját végrehajtási környezetet definiálhatunk, ahol a projekt buildjeit lehet különböző szcenáriók szerint futtatni. A végrehajtó típusa többek között SSH, Shell, VirtualBox, Kubernetes vagy Docker is lehet, amelyek közül a Shell Executor mellett döntöttem. A rendszer manuális elindításához korábban elegendő volt egyszerűbb docker és docker-compose parancsok megadása a szerver terminálján, emiatt a CI/CD folyamat során is elégséges a parancssori felület használata. Valamint amennyiben rendelkezésre áll, felhasználható a program korábbi klónozott verziója is. Ezt a változatot a legkönnyebb konfigurálni, emellett a szükséges függőségeket arra a szerverre kell feltelepíteni, amelyen a GitLab Runner is fut, nincs szükség bonyolultabb infrastruktúra kiépítésére [45].

6.3.4. A GitLab CI/CD változók

A GitLab CI/CD változók olyan környezeti változók, amelyek segítségével a pipeline-ok és az azokban definiált szintek (stages) különböző feladatainak (jobs) viselkedését képesek lehetünk vezérelni, valamint eltárolhatunk értékeket újrafelhasználás céljából, ezáltal elkerülhető, hogy a .gitlab-ci.yml fájlba legyenek beégetve akár érzékeny adatok és jelszavak. A GitLab elérhetővé tett számos előre definiált változót is, amelyeket szabadon felhasználhatunk a CI/CD pipeline kiépítése során. Emellett pedig saját, személyre szabott változókat is létrehozhatunk akár a YAML fájlban definiálva, vagy projekt, esetleg csoportszinten, de emellett a GitLab példányhoz is létrehozhatunk CI/CD változókat [46]. Az alábbiakban felsorolok néhány fontosabb, előre definiált változót, amelyeket a megvalósítás során alkalmaztam:

CI_REGISTRY: A GitLab Container Registry, azaz a konténer tárhely URL címét tartalmazza. A változó értelemszerűen csak akkor érhető el, ha a tárhely engedélyezve van a projekthez. A port értékét is tartalmazza abban az esetben, ha ez előzetesen meg lett adva a tárhely konfigurációja során.

CI_REGISTRY_IMAGE: A GitLab Container Registry, azaz a konténer tárhely URL címét tartalmazza. A változó értelemszerűen csak akkor érhető el, ha a tárhely engedélyezve van a projekthez.

CI_COMMIT_REF_SLUG: A branch vagy a címke neve, amelyen a projekt build létrejött. Értéke megegyezik a CI_COMMIT_REF_NAME változó kisbetűs, 63 bájtra rövidített megfelelőjével.

CI_COMMIT_SHA: A projekt build-hez tartozó commit revíziószáma.

CI_REGISTRY_USER: A GitLab projekt konténer tárhelyébe (Container Registry) történő feltöltéséhez szükséges felhasználónevet tartalmazza. Csak abban az esetben aktív, amennyiben a tárhely engedélyezve van a projekthez.

CI_REGISTRY_PASSWORD: A GitLab projekt konténer tárhelyébe (Container Registry) történő feltöltéséhez szükséges jelszót tartalmazza. Csak abban az esetben aktív, amennyiben a tárhely engedélyezve van a projekthez. A változó értéke megegyezik a CI_JOB_TOKEN-ével, és kizárólag a feladat (job) befejezéséig érvényes.

A projektszinten vagy feljebb létrehozott változók maszkolására is van lehetőség, amely előnyös lehet érzékeny adatok esetében, hiszen így nem kerül megjelenítésre a feladatok naplófájljaiban a folyamatos integráció során. Munkám során különös figyelmet kellett fordítanom az autentikációs adatok védelmére, illetve arra is, hogy egy újrafelhasználható és univerzális CI/CD pipeline-t építsek ki, emiatt a YAML fájlok nem tartalmazhatnak jelszavakat vagy speciális URL-eket.

6.4. Az implementáció során felmerült problémák és megoldásuk

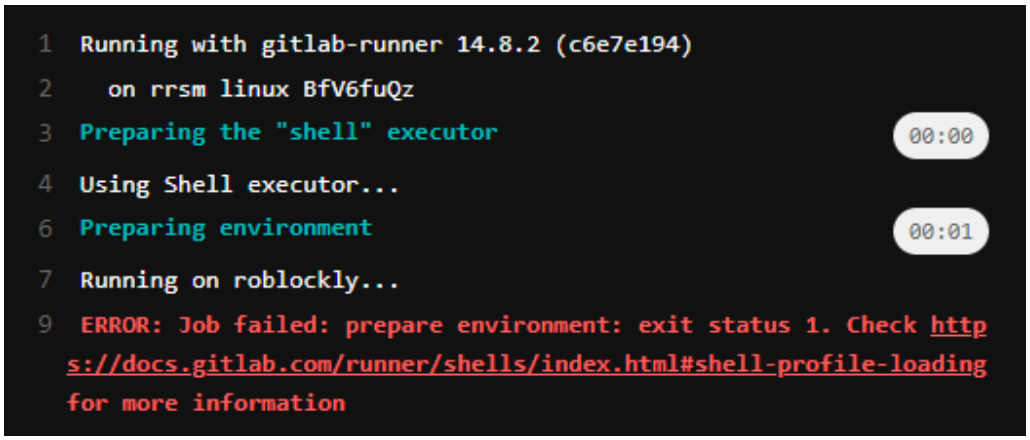
A teszt szerveren a CI/CD kialakítása közben a tesztelés és fejlesztés során számos olyan ideiglenes képfájl és konténer keletkezett, amelyre a későbbiekben nem lesz szükség, azonban a tárhelyen lévő területet foglalja, ami a szerver működésképtelenségéhez is vezethet. A .gitlab-ci.yml fájlban definiált parancs a törlésre azonban nem bizonyult megfelelő megoldásnak, mivel gyakran megakasztotta a CI/CD folyamatot. Emiatt egy naponta ütemezett cron parancs megadásával eszközöltem a problémát.

A Cron daemon egy beépített Linux segédprogram, amely ütemezi a rendszerben megadott folyamatokat a crontab (cron tábla) beolvasásával, amely az előre definiált parancsokat és szkripteket tartalmazza [47]. Ezáltal automatizálhatóvá válik számos folyamat, mint például a napi mentések kezelése, vagy akár a naplózási fájlok havonta történő archiválása is.

A crontab egy sorának alapértelmezett formája: a b c d e /directory/command output

A parancssorban megadott első öt mező (a-e) a feladat gyakoriságának idejét és dátumát határozza meg. A második rész, vagyis a /directory/command a szkriptet és annak elérési útvonalát definiálja, az output pedig egy opcionális szegmens, amely azt adja meg, hogy az adott felhasználó hogyan legyen értesítve a feladat elvégzését követően. A cron naplózási fájljai megtekinthetőek a var/log/cron mappában, amellyel ellenőrizhető a folyamat lefutása. A cron feladat beállításának lépéseit a Mellékletek fejezet 10.3. alfejezetében részletezem.

A CI/CD pipeline tesztelésekor egy másik hiba is kiütközött, ugyanis a Shell végrehajtó az Ubuntu 20.04-es verziójával nem kompatibilis, a környezetet nem tudta előkészíteni a feladat végrehajtásához.



6.6. ábra. A Shell executor hibája

A GitLab oldalán talált nyitott hibajegyre érkezett válaszok között megtaláltam a viszonylag egyszerűen implementálható megoldást erre. A /home/gitlab-runner/.bash_logout fájlban a konzol törlésére vonatkozó rész kikommentezése vált szükségessé, ezután a pipeline sikeresen lefutott [48].

```
cat .bash_logout
# ~/.bash_logout: executed by bash(1) when login shell exits.

# when leaving the console clear the screen to increase privacy

if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
```

6.7. ábra. A bash_logout fájl kikommentezése

7. Továbbfejlesztési lehetőségek

A CI/CD pipeline kiegészíthető automatizált tesztek futtatásával is, amely a kód szintaktikai ellenőrzése és validálása szempontjából hasznos funkció lehet a folyamatban lévő fejlesztések során. Ehhez elegendő a megfelelő .gitlab.ci.yml fájlban definiált testing szinten a szükséges parancsok megadása. A Docker képfájlok alaposabb felülvizsgálása optimalizációs szempontból hasznos lehet (kisebb alap képfájlok választása, a telepített csomagokból kizárólag a szükségesek megtartása, a cache kihasználása és a rétegek számának csökkentése), ezáltal a buildelés és a telepítés folyamata gyorsabbá válhat.

A skálázódás stabilizálásának megtervezésével és a szűk keresztmetszetek beazonosításával a valós idejű kapszulából több példány egyidejű, biztonságos futása lehetőséget teremthet akár egy többfelhasználós platform megvalósítására is.

Továbbá a Roblockly szoftver abból a szempontból is megvizsgálható, hogy a felhasználók mennyire hatékonyan és eredményesen képesek használni a korábban megfogalmazott célokra. Ehhez egy usability teszt elvégzése egy fókuszcsoport bevonásával elégséges lehet, amely során a tanulhatóság, megjegyezhetőség és a program hatékonysága a fő vizsgálati irányzat. Ezek a kérdések főképp a szoftver kezelésének tanulhatóságát és megjegyezhetőségét, illetve a program hatékonyságát, hibatűrését és a felhasználói elégedettséget mérik fel.

8. Összegzés

Szakdolgozatomban bemutatásra került a Roblockly szoftver, amely a blokk alapú programozói felület megvalósításával oktatási vagy ipari alkalmazásra is ideális lehet, ezt alátámasztja az alkalmazott technológiákról szóló szakirodalom, illetve a számos hasonló platform megjelenése is a piacon. A MaxWhere virtuális valóság környezet adottságainak kihasználásával egy igazán kiemelkedő műszaki lehetőséggé válhat a szakterületen.

Vállalati szempontból a projekt életciklus menedzsmentjének megfelelő kialakításához kiemelten fontosak a jól megtervezett fejlesztési fázisok, amelyek a kód megírásától a tesztek futtatásán át a folyamatos szállítás és üzembe helyezés lépéseit is magukban foglalják. Az egyes lépések pontos dokumentálása és a verziókövetés a reprodukálhatóság, illetve az üzleti folytonosság megőrzése érdekében elengedhetetlen alappillére a szoftverfejlesztésnek. Munkám során ennek fontosságát igyekeztem kiemelni, valamint törekedtem egy megbízható szisztéma kialakítására a CI/CD pipeline megvalósításával, amely megkönnyítheti a projekten dolgozó fejlesztők mindennapi munkáját.

9. Köszönetnyilvánítás

Szeretnék köszönetet mondani konzulenseimnek, Dr. Galambos Péternek és Budai Tamásnak, hogy szakmai tudásukkal és tapasztalatukkal támogattak a szakdolgozatom elkészítésében, valamint Tarsoly Sándornak, aki a megvalósítás folyamatát végigkövetve számos jótanáccsal és segítséggel látott el.

Nagyon hálás vagyok továbbá családomnak és barátaimnak mindazon támogatásért és megértésért, amit ez idő alatt irányomba mutattak, megkönnyítve ezzel is a mindennapjaimat egy kihívásokkal teli időszakban.

Hivatkozások

[1] S. Chaturvedi, „So what exactly is 'deep technology'?." Available at https://www.linkedin.com/pulse/so-what-exactly-deep-technology-swati-chaturvedi/ (2022/05/08).

[2] S. Vaidyaa, P. Ambadb, and S. Bhoslec, „Industry 4.0 – a glimpse," in *Manufacturing Letters*, 2nd International Conference on Materials Manufacturing and Design Engineering, pp. 233–238, Elsevier B.V., 2018.

[3] D. P. Skobelev and D. S. Borovik, „On the way from industry 4.0 to industry 5.0: From digital manufacturing to digital society," *International Scientific Journal*, vol. 2, no. 6, pp. 307–311, 2017.

[4] G. Csapó, „Vizuális programozás oktatása középiskolákban," Master's thesis, Faculty of Informatics, University of Debrecen, 2016.

[5] C. Gábor and S. Katalin, „Oktatóprogram a sprego táblázatkezelő módszerhez," in *Informatika Szakmódszertani Konferencia*, INFODIDACT 2015, pp. –, Webdidaktika az Oktatásért és az Információs Társadalomért Alapítvány, 2015.

[6] „Behavioral programming." Available at https://www.wisdom.weizmann.ac.il/~bprogram/index.html (2022/05/13).

[7] G. Sayfan, „Introduction to event-based programming." Available at https://aiven.io/blog/introduction-to-event-based-programming (2022/05/13).

[8] Y. Rao, „What is ... flow-based programming?." Available at https://developer.ibm.com/videos/what-is-flow-based-programming/ (2022/05/13).

[9] S. Morgan, „Simulating the world with microsoft robotics studio." Available at https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/june/robotics-simulating-the-world-with-microsoft-robotics-studio (2022/05/13).

[10] „Tp programming." Available at https://www.onerobotics.com/tags/tp-programming/ (2022/05/05).

[11] „Introduction to karel programming.” Available at https://www.onerobotics.com/posts/2013/introduction-to-karel-programming/ (2022/05/05).

[12] A. Owen-Hill, „What is the best programming language for robotics?.” Available at https://blog.robotiq.com/what-is-the-best-programming-language-for-robotics (2021/04/19).

[13] „Kuka sunrise.os.” Available at https://www.kuka.com/en-hu/products/robotics-systems/software/system-software/sunriseos (2022/05/05).

[14] „How to use moveit to develop a robotic manipulation application.” Available at https://robotnik.eu/moveit-manipulation-application/ (2022/05/05).

[15] Robotnik, „Moveit related projects.” Available at https://moveit.ros.org/documentation/related_projects/ (2022/05/05).

[16] A. Owen-Hill, „What is the best way to program a robot?.” Available at https://robodk.com/blog/program-robot-tips/ (2021/04/19).

[17] A. Kantarci, „Low/ no code development: What it is, how it works, use cases.” Available at https://research.aimultiple.com/low-code/ (2021/04/19).

[18] „ABB - robotstudio.” Available at https://new.abb.com/products/robotics/robotstudio/ (2021/04/19).

[19] A. Robotics, *Technical reference manual RAPID Instructions, Functions and Data types*. 2004.

[20] „Robot control mate.” Available at https://new.abb.com/products/robotics/robotstudio/robot-control-mate (2022/05/13).

[21] „ABB - wizard easy programming.” Available at https://new.abb.com/products/robotics/application-software/wizard (2022/04/28).

[22] „What are the benefits of roxy?.” Available at https://roxy-robotics.com/en/ (2021/04/19).

[23] „Ready robotics - apps for next-level automation.” Available at https://www.ready-robotics.com/products/ready-apps (2021/04/19).

[24] „drag&bot os: graphical robot operating system.” Available at https://www.dragandbot.com/software/ (2021/04/19).

[25] J. Tobik, „Ipari robotok web-alapú generikus programozása,” Master's thesis, Neumann Faculty of Informatics, Óbuda University, 2020.

[26] „Javascript.” Available at https://developer.mozilla.org/hu/docs/Web/JavaScript (2021/04/19).

[27] „Maxwhere.” Available at https://www.maxwhere.com/ (2021/04/19).

[28] „Node.js.” Available at https://en.wikipedia.org/wiki/Node.js (2021/04/19).

[29] „The standard ros javascript library.” Available at http://wiki.ros.org/roslibjs (2021/04/19).

[30] „Express web framework (node.js/javascript).” Available at https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs (2021/04/19).

[31] „Vue.js.” Available at https://vuejs.org/v2/guide/ (2021/04/19).

[32] „React.” Available at https://hu.reactjs.org/ (2021/05/13).

[33] „Bootstrap.” Available at https://getbootstrap.com/ (2021/04/19).

[34] „Blockly.” Available at https://developers.google.com/blockly/guides/get-started/web (2021/04/19).

[35] „Traefik.” Available at https://doc.traefik.io/traefik/ (2022/03/27).

[36] „Docker.” Available at https://docs.docker.com/get-started/overview/ (2021/04/19).

[37] „Docker-compose.” Available at https://docs.docker.com/compose/ (2022/02/27).

[38] „Aws documentation.” Available at https://docs.aws.amazon.com/index.html?nc2=h_mo (2021/04/19).

[39] „Cicd.” Available at https://docs.gitlab.com/ee/ci/ (2022/03/29).

[40] „The .gitlab-ci.yml file.” Available at https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html (2022/05/02).

[41] „Multi-project pipelines.” Available at https://docs.gitlab.com/ee/ci/pipelines/multi_project_pipelines.html (2022/05/13).

[42] „Gitlab registry.” Available at https://docs.gitlab.com/ee/user/packages/container_registry/ (2022/04/12).

[43] „Gitlab runner.” Available at https://docs.gitlab.com/runner/ (2022/04/12).

[44] „Registering runners.” Available at https://docs.gitlab.com/runner/register/index.html (2022/05/10).

[45] „Gitlab executor.” Available at https://docs.gitlab.com/runner/executors/ (2022/04/12).

[46] „Gitlab variables.” Available at https://docs.gitlab.com/ee/ci/variables/ (2022/04/12).

[47] „How to set up a cron job in linux.” Available at https://phoenixnap.com/kb/set-up-cron-job-linux (2022/05/10).

[48] „Shell executor fails to prepare environment in ubuntu 20.04.” Available at https://gitlab.com/gitlab-org/gitlab-runner/-/issues/26605 (2022/05/03).

[49] D. Ocean, „How to install and use docker on ubuntu 20.04.” Available at https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04 (2022/05/02).

[50] D. Ocean, „How to install and use docker compose on ubuntu 20.04.” Available at https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04 (2022/05/02).

Ábrajegyzék

3.1. Wizard Easy Programming szoftver 12
3.2. Az ABB FlexPendant felülete 13
3.3. Az IRB 1100-as ipari robotkar 14
3.4. A YuMi robotkar 15
3.5. A Roxy Robotics felülete 16
3.6. A Ready Robotics felülete 17
3.7. A drag&bot felülete 18
4.1. A Roblockly webalkalmazás felülete 19
4.2. A MaxWhere környezet 20
4.3. A Roblockly alkalmazás architektúrája 22
4.4. A kapszula felépítése 24
4.5. Blockly kódgenerátor 27
4.6. A komponensek közötti kommunikáció 29
4.7. NginX definiálása a docker-compose.yml fájlban 33
5.1. Hiba az RRSM image buildelése során 35
5.2. A Docker Service fájl 37
5.3. A docker-compose.yml fájl módosítása 37
6.1. A CI/CD folyamatok 40
6.2. A multi-projekt pipeline kialakítása 42
6.3. A multi-projekt pipeline 43
6.4. A GitLab konténer tárhelye 44
6.5. Az RRSM projekthez tartozó specifikus GitLab Runner 45
6.6. A Shell executor hibája 48
6.7. A bash_logout fájl kikommentezése 48

10. Mellékletek, felhasználói és fejlesztői dokumentációk

10.1. A lokális környezet kialakítása teszteléshez virtuális gépen

10.1.1. Ajánlott rendszer előkövetelmények

- Operációs rendszerek emulációjára és virtualizálására alkalmas szoftver, mint például az Oracle VirtualBox vagy a VMware telepítése
- Ubuntu 20.04-es operációs rendszer telepítése virtuális gépre
- A titkosított fájlátvitelhez és a szerver távoli SSH eléréséhez terminálemulátor vagy SFTP kliens, úgymint a PuTTY vagy a WinSCP telepítése

10.1.2. A Docker telepítése

- sudo apt update: csomagok frissítése a szerveren
- sudo apt install apt-transport-https ca-certificates curl software-properties-common: az APT csomagkezelő HTTPS kapcsolat használatához szükséges csomagok letöltése
- curl -fsSL https://download.docker.com/linux/ubuntu/gpg — sudo apt-key add -: a hivatalos Docker adattár GPG kulcsának hozzáadása a rendszerhez
- sudo add-apt-repository ”deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable”: a Docker adattár hozzáadása az APT forrásokhoz. Ez a csomag adatbázist is frissíti az újonnan hozzáadott Docker csomagokkal, ezért ellenőrizni kell, hogy a Docker adattárat használjuk az alapértelmezett Ubuntu helyett a következő paranccsal. Fontos, hogy az url-ben szereplő verziószámot leellenőrizzük, és a legfrissebbre hivatkozzunk vele
- apt-cache policy docker-ce: az apt-cache szabályzat ellenőrzése a docker-ce verziójára vonatkozóan
- sudo apt install docker-ce: a Docker telepítése
- sudo systemctl status docker: a Docker futásának ellenőrzése

- sudo usermod -aG docker $USER: opcionálisan a felhasználó hozzáadása a docker csoporthoz. Ezáltal a docker parancsok a sudo beírása nélkül is használhatóak nem-root felhasználók számára is.
- su - $USER: az új beállítások alkalmazásához meg kell adni ezt a parancsot, vagy ki kell jelentkezni majd újból belépni a szerverre
- groups: a felhasználó csoportjainak ellenőrzése

Az installáláshoz további segítséget a DigitalOcean oldalán, vagy számos más oktatóanyag felkeresésével kaphatunk [49].

10.1.3. A Docker-Compose telepítése

- sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose: a jelenlegi compose fájl nem kompatibilis a 2.xx docker-compose verzióval, csak a 1.29.2-vel, emiatt a verziószámot itt nem kell módosítani
- sudo chmod +x /usr/local/bin/docker-compose: a docker-compose parancs futtathatóvá tétele
- docker-compose –version: a telepítés ellenőrzése

A telepítéshez a DigitalOcean oldalán lévő oktatóanyagot vettem alapul [50].

10.1.4. Docker képfájlok létrehozása

A konténerek képfájljainak létrehozására többféle módszert alkalmazhatunk:

- docker build [dockerfile] -t [filepath] build image: ezzel a paranccsal a meglévő Dockerfile használatával buildeljük a képfájlt (például: docker build -t rrsm .)
- docker load [OPTIONS]: archivált tar kiterjesztésű fájlból való betöltés (például: docker load capsule.tar)
- docker pull [name]: adattárból való letöltés (például: docker pull gitlab.maxwhere.com:5050/roblockly/roblockly)

10.1.5. Docker hálózat létrehozása

- docker network create [OPTIONS] NETWORK: a RoblocklyNet létrehozása a konténerek kommunikációjához (docker network create RoblocklyNet)

10.1.6. A rendszer elindítása docker-compose segítségével

- A docker-compose.yml fájl átmásolása a szerverre
- docker-compose up -d: a felkonfigurált docker-compose.yml fájl mappájában a parancs segítségével elindítjuk a rendszert. A -d vagy -detached opció azt jelenti, hogy a konténerek a háttérben futnak, és a konzolt továbbra is használhatjuk

10.1.7. A host portszámának elérhetővé tétele docker konténerből

- sudo nano /lib/systemd/system/docker.service: a docker.service megnyitása szerkesztésre
- -H tcp://172.17.0.1: a host IP címének megadása a Docker serviceben a következő sorban: ExecStart=/usr/bin/dockerd
- sudo systemctl daemon-reload: a docker daemon újratöltése
- sudo systemctl restart docker: a docker újraindítása

A docker.service konfigurációját érdemes ellenőrizni a szerver újraindítását követően, mert előfordulhat, hogy az alapértelmezett beállítások kerülnek betöltésre, és a fenti lépéseket ismételten el kell végezni.

10.2. A GitLab CI/CD pipeline-hoz szükséges eszközök telepítése és konfigurálása

10.2.1. A GitLab Runner telepítése

A GitLab Runner letöltése és telepítése lehetséges egy bináris fájl használatával, vagy a deb vagy az rpm csomagkezelő segítségével is. Munkám során az első módszert

használtam, így ennek lépései kerülnek ismertetésre, azonban a GitLab dokumentációban bemutatásra kerül a második megközelítés is.

A különböző rendszerekhez különböző verziójú bináris állomány letöltése szükséges, ebben az esetben a Linux arm64-es verzióhoz tartozót választottam:

- sudo curl -L –output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-arm64
- sudo chmod +x /usr/local/bin/gitlab-runner: a GitLab runner futtathatóvá tétele
- sudo useradd –comment 'GitLab Runner' –create-home gitlab-runner –shell /bin/bash: GitLab CI felhasználó létrehozása
- sudo gitlab-runner install –user=gitlab-runner –working-directory=/home/gitlab-runner: a runner installálása
- sudo gitlab-runner start: a runner indítása

10.2.2. A GitLab Runner regisztrálása

- sudo gitlab-runner register: a regisztráció indítása a szerveren. A következőkben a regisztráció során megadandó paraméterek kerülnek részletezésre.
- A GitLab példány URL címének megadása. Mivel a GitLab beépített CI/CD eszközét használjuk, az alapértelmezett címet kell megadnunk: https://gitlab.maxwhere.com/
- A runner regisztrálásához kapott token megadása, amely a GitLab projekt menüjében a Settings/CICD/Runners/Specific Runners menüpontban található
- Opcionálisan megadható egy leírás a runner-hez, amelyet a GitLab felhasználói felületén lehet módosítani a korábban említett menüpontban
- A runnerhez tartozó címkék megadása vesszővel elválasztva, amelyeket szintén lehet módosítani a korábban említett menüpontban. Ezek definiálása azért fontos, hogy a .gitlab-ci.yml fájlban definiált feladatokat a runner megtalálja és lefuttassa.
- A runnerhez tartozó opcionális, a kezeléssel kapcsolatos megjegyzés megadása

- A runnerhez tartozó végrehajtó megadása. A jelenlegi felhasználási esethez a shell megadása szükséges.

A GitLab Runner regisztrálásához a GitLab dokumentáció Linux rendszerre vonatkozó részét vettem alapul [44].

10.3. A Cron segédprogram konfigurációja

10.3.1. Előkövetelmények

- Linux operációs rendszer
- hozzáférés a terminálhoz
- felhasználó rendszergazda jogosultsággal

10.3.2. A Cron feladat beállítása

Mivel a gitlab-runner felhasználó a CI/CD pipeline működéséhez elengedhetetlen, ezért ehhez a felhasználóhoz állítottam be az ütemezett feladatot. Ehhez az alábbi parancsok futtatására van szükségünk:

- sudo crontab –u gitlab-runner –e: a crontab szerkesztése a gitlab-runner felhasználónak
- 0 0 * * * docker system prune -f >/dev/null 2>&1: az ütemezett feladat naponta, éjfélkor kerül lefutásra, vagyis a docker törli a nem használt konténereit, hálózatait és képfájljait. A kimenet opcionális, alapértelmezetten a Cron emailt küld a felhasználó számára, amikor a parancs lefut, azonban erre jelen helyzetben nincs szükség, ezért a fenti paraméter megadásával ez letiltásra kerül.
- sudo crontab -u gitlab-runner –l: ezzel a paranccsal listázhatóak a gitlab-runner cron feladatai.

A Cron feladat beállításához a megjelölt forrást vettem alapul [47].